JANUARY 1991 £2.75

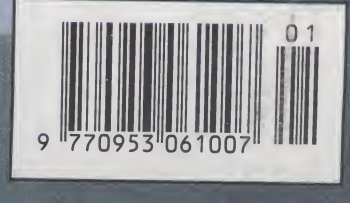
Commodore DISK USER

2X2 CHARACTER EDITOR
Larger Fonts Made Easy

CBMPC/LINK REVIEWED

PROBLEM SOLVING

ADVENTURE HELPLINE



STAR PRINTERS

	EX VAT	INC VAT
LC10 and LC10-C Superb value printers! 120CPS draft, 30CPS NLQ Multiple fonts	£139	£159.95
LC10 COLOUR The above specification plus full colour	£190	£218.50
LC24-10 24 pin 142 CPS draft. 47 CPS NLQ	£230	£284.50

SPECIAL OFFERS

AMIGA A500 BATMAN PACK OR FLIGHT OF FANTASY PACK

* DELUXE PAINT II
* INTERCEPTOR
* NEW ZEALAND STORY
* BATMAN
* MOUSE
* MODULATOR

PLUS

* 5 EXCITING ORIGINAL GAMES
* JOYSTICK
* 10 3 1/2 INCH DISKS
* DISK BOX

EX VAT 346.75 INC £399

PANASONIC PRINTERS

	EX VAT	INC VAT
KXP-1081 Outstanding value! 120 CPS draft 24 CPS NLQ. Friction and tractor feed.	£129	£148.35
KXP-1180 192 CPS draft, 38 CPS NLQ	£180	£184.00
KXP-1124 24 pin 192 CPS draft 38 CPS NLQ	£240	£276.00

SEIKOSHA PRINTERS

	EX VAT	INC VAT
SP-200 190 CPS draft 40 NLQ	£173	£199
SL-92 24 pin 240 CPS draft 80 CPS NLQ	£271.30	£312
SL-1350	£313.00	£360
SL-1350	£433.90	£640
BP-5500	£1169.56	£1345.00

ACE COMPUTERS

MICRO SPECIALIST

BRANCHES THROUGHOUT THE UK

FOR MAIL ORDER CALL

VISA Access

0272 663312

HEAD OFFICE
30 Cannon Street, Bedminster,
BRISTOL BS3 1BN.
Tel: 0272 663312
Fax: 0272 231455

HOME COMPUTERS AND EQUIPMENT REPAIRS
3 North Street, Bedminster,
BRISTOL.
Tel: 0272 666341

87 City Road,
CARDIFF.
Tel: 0222 483069/471600

54 Commercial Road,
NEWPORT (Gwent).
Tel: 0633 212176

16 West Street,
WESTON-SUPER-MARE.
Tel: 0934 419040

31 Farringdon Road,
SWINDON.
Tel: 0793 512073/4

38 Robertson Street,
HASTINGS.
Tel: 0424 422929

92 Upper Parliament Street
NOTTINGHAM
Tel: 0602 473873

3/5 Samboun Road
WARMINSTER
Tel: 0985 846131

ALL PRICES

are subject to alteration without notice and

INCLUDE FREE NEXT DAY DELIVERY

within mainland Great Britain

ACE SERVICES TO BUSINESS

Ace Computers are able to advise their business customers over the whole range of computer needs. Our services include analysing the problem and recommending computer solutions; supplying computer systems including all equipment and programmes; installation of equipment and training.

We offer a range of financing facilities – written details on request – and our maintenance contracts guarantee next day service throughout the country.

Whether you are computerising for the first time, or updating existing systems, ACE will be most pleased to hear from you!

Call our Customer Liaison Desk at the nearest shop, or on our Bristol number.

SPECIAL MESSAGE TO RETAILERS

ACE are able to supply EPOS and computerised stock control systems to suit from one to eight shops. These systems can revolutionise your work load and substantially reduce the costs of financing stock.

Our comprehensive service includes programme modification to meet your needs; supply and installation of equipment and training.

Financing facilities are available – written details on request – and our maintenance contracts guarantee next day service.

Prices will surprise you! Call ournearest shop or our Bristol number and ask for the Customer Liaison Desk for more details.

FRANCHISE SERVICES

Ace Computers have a highly successful franchise operation. If you are interested in opening your own comuter shop, please call us and ask for a copy of the Franchise Brochure.

WE HAVE A NUMBER OF SHOPS READY TO START IN BRISTOL AND NOTTINGHAM

PHILIPS MONITORS

	EX VAT	INC VAT
7502 Mono Leads included	£78	£87.79
8833 Colour Leads inc. ST & Amiga compatible	£225	£255.75

3 1/2" HIGH QUALITY DS/DD DISKS INC. FREE BOX FOR £9.50

CONTENTS

Commodore DISK·USER·
Volume 4 Number 3 JANUARY 1991

ON THE DISK

IN THE MAGAZINE

Publisher: Hasnain Walji
Group Editor: Paul Eves
Technical Editor: Jason Finch
Publishing Consultant: Paul Crowder
Advertisement Manager: Cass Gilroy
Designer: Mark Newton
Distribution: Seymour Press Distribution
Ltd. Winsor House, 1270 London Road, Norbury, London
SW16 4DH. Tel:081 679 1899. Fax: 081 679 8907
Printed By: Gibbons Barford Print

Subscription Rates

UK	£33.00
Europe	£39.00
Middle East	£39.30
Far East	£41.60
Rest of World	£39.70 or $69.00

Airmail rates on request
Contact: Select Subscriptions. Tel: (0442) 876661

Commodore Disk User is a monthly magazine published on the 3rd Friday of every month. Alphavite Publications Limited, 20, Potters Lane, Kiln Farm, Milton Keynes, MK11 3HF. Telephone: (0908) 569819 FAX: (0908) 260229. For advertising ring (0908) 569819

 ISSN 0953-0614

f/s/s/i
COMPUTER SOFTWARE

SUPER SNAPSHOT v5

Well, it may have taken a few years of hard work and five powerful versions, each one breaking new ground, but Super Snapshot has become the best cartridge in the world. The list below details the main features of Super Snapshot v5; if you need a little more persuasion look back to CDU issue 19, you'll be impressed.
So take a look at the red box you've got plugged in, and if our specifications knock it for six or you don't own a cartridge then don't just sit there, buy Super Snapshot v5 today!

FEATURES:

- ▶ All features available at the press of a button
- ▶ Works with all 64 (c) and 128 (D) computers
- ▶ Compatible with 1700/1764/1750 REU'S
- ▶ Snap any memory resident program into 1 file
- ▶ Save 7× faster and load 15× faster on the 1541, 1571 & 1581. Speeds of up to 25× faster when using Turbo 25 – even faster than Replay
- ▶ Super DOS Wedge
- ▶ GAME MASTER menu with sprite killer, infinitive lives generator and joystick port swapper
- ▶ Programmable function Keys
- ▶ Sprite Monitor
- ▶ Exclusive Character Set Monitor
- ▶ Exclusive Sound Sample Monitor
- ▶ Exclusive Boot sector support
- ▶ 300/1200/2400 Terminal program (40/80 column)
- ▶ SUPER DISK SNAPSHOT – our new super nibbler
- ▶ SCREEN-COPY now loads or saves in more formats and dumps in COLOUR to STAR LC10C printers and in 16 grey scales
- ▶ Improved full featured M/L monitor that DOES NOT CORRUPT MEMORY. Interrupt, examine and resume any running program
- ▶ Drive Mon
- ▶ BASIC PLUS with 15 new BASIC commands
- ▶ FILE MANAGEMENT SYSTEM – scratch, unscratch, rename or adjust skew. Includes our 1 or 2 drive file copier with partition support for the 1581
- ▶ Fast disk copier, 1 or 2 drives
- ▶ OUR FILE COPIER, DISK COPIERS and NIBBLER MAKE FULL USE OF THE REU'S
- ▶ Sequential file reader
- ▶ Utility disk
- ▶ Plus 150+ Kracker Jax parameters

ONLY £34.95

ANIMATION STATION

Unleash your creativity with Animation Station, a powerful utility for generating all kinds of graphics on your Commodore 64 or 128. Built-in, predrawn pictures give you a head start on your creations. Automatic generation of circles, ovals, squares, boxes, straight lines, typography and other geometric shapes gives you all the tools of a self-contained, electronic drafting room. Combine type and graphics on the screen, draw in many colours, even connect your VCR to create titles and graphics for your home movies. Screen dump to your printer. Koala Compatible. Package includes design touch pad, pencil and graphics software. Ideal for GEOS users.

ONLY £59.95

VIDEO BYTE 3

Digitize video images from your VCR, laser disk, B/W or colour camera, off the air or cable TV. New version 3.0 software features full re-display with multi-capture mode, menu select printing, expanded colourizing features, save to disk and much more. The hardware is no larger than an average cartridge which plugs into the User Port. The menu driven capture software is easy to use and pictures stored to disk can be imported into most popular drawing packages including GEOS. Prints in full colour to the Star LC10 when used with SuperSnapshot.

ONLY £79.95

HOME VIDEO PRODUCER 64

Nothing can make home videos so special. Add titles, text, and even brilliant graphics to your favourite home videos with ease and the help of the Home Video Producer. You have the choice of 10 typefaces, 75 large full-colour graphics and ready made segments, but the most appealing aspect of the Home Video Producer is the ease with which you can do all this.

ONLY £29.95

C64 SLIMLINE UPGRADE CASE

Make your older C64 look like a newer model! All you need is a screwdriver and about 15 minutes to transfer your C64 insides to this new case. Complete instructions included.

ONLY £12.95

CARTRIDGE PORT EXTENDER CABLE

Are you cramped for space behind your computer? Is it hard to reach your cartridge port to plug-in or swap cartridges? This handy cable is the solution. The Cartridge Port Extender Cable connects to the cartridge port in the back of the computer and lets you plug in your cartridge to its other end. Since the cable is flexible, you can locate the cartridge up to 11" away for easier access. Not for REU's.

ONLY £19.95

GEOS APPLICATIONS

GEOSCAN ART

This special type of GEOS art has been created using The HandyScanner 64. Pictures are scanned at 400 dpi from magazines, books and papers to create the first geoSCAN ART Collection entitled The British Countryside and is packed full with Eagles, Owls and Butterflies.

ONLY £6.95

GEODIRECTORY

A comprehensive book listing all available GEOS programs. Details for each program is given including version numbers. The geoDirectory is divided into sections covering Paint, Write, Spell, File, Calc, Chart, Terminal, Graphic, Music, Animation, Games and many more.

ONLY £6.95

GEOTRONIX

A professional PCB designer utilizing the GEOS environment. Five double sided disks supply geoPublish with pre-designed components, sockets, edge connectors and layout grids in Photo Scraps. Using the Photo Manager and geoPublish the circuit is designed and printed.

ONLY £39.95

GEOS 64 V2.0	£29.95
GEOCALC 64	£24.95
GEOFILE 64	£24.95
GEOPROGRAMMER	£29.95
DeskPack Plus	£19.95
GEOCHART	£19.95
GEOS 128 V2.0	£39.95
GEOCALC 128	£29.95
GEOFILE 128	£29.95
GEOPUBLISH	£29.95
FontPack Plus	£19.95
Int. Font Pack	£19.95

VISA Access

Prices include VAT & U.K. delivery. Overseas orders send advertised price plus £2.50 for Airmail. Please send cheque, Postal Order or Credit Card details. VISA/ACCESS orders accepted by 'phone.

**Masons Ryde,
Defford Road,
Pershore, Worcs.
WR10 1AZ.
Tel: (0386) 553153
Technical Support
Tel: (0386) 553222**

MERRY CHRISTMAS and HAPPY NEW YEAR

It doesn't seem possible does it? Why, only a few months ago, or so it seems, I was wishing all of you MERRY CHRISTMAS and HAPPY NEW YEAR. So once again, I echo those words of last year. I sincerely hope that each and everyone of you has the kind of festive season that they are hoping for.

This last twelve months has seen a lot of changes in this world, and even more at the CDU offices. A lot of them you are all aware of (but there are just as many that you aren't). One of those changes you will notice on this months disk. FOR THIS ISSUE ONLY, you will have to load the menu with the command LOAD"CDU MENU",8,1 and not the normal LOAD"MENU",8,1. This is because of the way one of the programs has been programmed on the disk.

On the disk is a program called TERMINUS. This program is in fact one of those DEMOs that I am constantly being asked to publish. HOWEVER, this one is decidedly different to all the normal demos you see. Enjoy!!

I will finish now by saying once again, a very big thank you for your past interest and support of the magazine. Have a really great Christmas. See you all in the new year.....Paul Eves (El Groupo!!).

DISK INSTRUCTIONS

Although we do everything possible to ensure that CDU is compatible with all C64 and C128 computers, one point we must make clear is this. The use of 'Fast Loaders', 'Cartridges' or alternative operating systems such as 'Dolphin DOS', may not guarantee that your disk will function properly. If you experience problems and you have one of the above, then we suggest you disable them and use the computer under normal, standard conditions. Getting the programs up and running should not present you with any difficulties, simply put your disk in the drive and enter the command.

LOAD"CDU MENU",8,1

Once the disk menu has loaded you will be able to start any of the programs simply be selecting the desired one from the list. It is possible for some programs to alter the computers memory so that you will not be able to LOAD programs from the menu correctly until you reset the machine. We therefore suggest that you turn your computer off and then on again, before loading each program.

HOW TO COPY CDU FILES

You are welcome to make as many of your own copies of CDU programs as you want, as long as you do not pass them on to other people, or worse, sell them for profit. For people who want to make legitimate copies, we have provided a very simple machine code file copier. To use it, simply select the item FILE COPIER from the main menu. Instructions are presented on screen.

DISK FAILURE

If for any reason the disk with your copy of CDU will not work on your system then please carefully re-read the operating instructions in the magazine. If you still experience problems then:

1. If you are a subscriber, return it to:
Select Subscriptions Ltd
5, River Park Estate
Berkhamsted
Herts
HP4 1HL
Telephone; 0442 876661

2. If you bought it from a newsagents, then return it to:
CDU Replacements
Interceptor Group
Mercury House
Calleva Park
Aldermaston
Berks
RG7 4QW
Telephone; 0734 817421

Within eight weeks of publication date disks are replaced free.

After eight weeks a replacement disk can be supplied from INTERCEPTOR GROUP for a service charge of £1.00. Return the faulty disk with a cheque or postal order made out to INTERCEPTOR GROUP and clearly state the issue of CDU that you require. No documentation will be supplied.

Please use appropriate packaging, cardboard stiffener at least, when returning disk. Do not send back your magazine, only the disk please.

NOTE: Do not send your disks back to the above address if its a program that does not appear to work. Only if the DISK is faulty. Program faults should be sent to: BUG FINDERS, CDU, Alphavite Publications Ltd, Unit 20, Potters Lane, Kiln Farm, Milton Keynes, MK11 3HF. Thank you.

2 by 2 CHARACTER EDITOR

Make large fonts, 2 characters wide by 2 characters deep with ease.

ROBERT TROUGHTON

This new character editor is specifically designed for making large fonts that are 2 characters wide and 2 characters deep. It is usual for the fonts to be stored in the format as illustrated below;

chr+$00	**chr+$40**
chr+$80	**chr+$C0**

So for the letter 'A' (poke code = $01) we would put on screen the following format;

$01	**$41**
$81	**$C1**

USES OF THE UTILITY

Originally the utility was written specifically for making character sets to be used in demos. This version feature no facilities for making hires (single colour) character sets, but that is because I will be writing a totally different editor for hires fonts shortly. Other than demos, the editor can also be used for many other projects...Text Pages, Games, Score Tables, Title Screens to name but a few.

WHAT IT CAN DO

This editor features 2 UNIQUE features, a 'Pattern Filler' and a '1*1 Character Expander', integrated into the font editor. All the usual commands are catered for also. Joystick or Cursor control, scroll character in 4 directions, flip character left, right, top or bottom, reverse character, clear character, copy character, swap colours and of course all the necessary disk functions.

THE PATTERN FILLER

The best way to learn how to use this facility is to practice. If you have designed a whole 'blank-filled' character set (see next paragraph) ALWAYS save it disk, if you make a mistake in your patterns, you can then reload the set and fix your problem.

A blank-filled character set is one which, quite obviously, has no fill pattern on it. There are 4 pattern designs, each of these will replace the colours in you 'blank-filled' character set. The 1st patter design replaces the black (ie; colour 0), the 2nd replaces colour 1, the 3rd replaces colour 2 and the 4th pattern will replace colour 3. It is usual to leave the 1st pattern blank, so press the '+' key to skip it.

By using the pattern filler, you can create lots more character sets which look very different from each other, but all using the same base. Just change the patterns and re-fill the 'blank-filled' character set! DO NOT try filling an already filled character set because the result will be very strange.

MAIN EDITOR

KEY CONTROLS (or use JOYSTICK in port 2 to control cursor)

* or FIRE	Set bit
SPACE	Clear bit
0,1,2,3	Select colours
SHIFT 1,2,3	Alter colours
U,D,L,R	Scroll (4 directions)
X,Y	flip left to right, top to bottom
CTRL R	Reverse current character
CBM R	Reverse whole character set
CBM C	Clear whole character set
CLR	Clear current character
C	Copy character
S	Swap colours
F3	Disk menu
F5	Pattern fill menu

DISK MENU

L	Load character set
S	Save character set
Z	Load 1*1 character set + expand to 2*2
D	Directory
F5	Pattern fill menu
F7	Main Editor

PATTERN FILL MENU

L	Load fill patterns
S	Save fill patterns
E	Edit fill patterns
D	Directory
F	Pattern fill character set
F3	Disk menu
F7	Main editor

NOTE: Edit fill patterns has only some of the functions from the main editor.

As with most utilities of this kind, the best way of learning is to practice and experiment to your hearts content.

ULTIMATE PASSWORD PROTECTION

Another program protection routine gets a viewing for all and sundry to enjoy

ROBERT TROUGHTON

Imagine that you have a program that you don't want anybody else to use. What do you do to prevent them from accessing it? Simple - you add a 'password' check to the front of the program.

In this short article I intend to reveal a form of password protection that even the most hardened hacker would have trouble in cracking! (In fact, it would be very near to impossible to crack!)

PASSWORDS

Everybody must know what a password is, but how can they be used? The most simple form of password-protection would look something like this BASIC listing:

```
10   INPUT "ENTER PASSWORD";A$
20   IF A$ <> "KIPPERMAN" THEN NEW
30   PRINT "PASSWORD ACCEPTED"
40   ... rest of program
```

It wouldn't take much effort (from anybody) to 'crack' the password from that program! Even by writing the above program in Machine Code, the program would still be very easy to be broken into!

ENCRYPTION

Encryption is a way of making code look RANDOM. Take the code below (machine code listed as hex):-

```
: A9 00 8D 20 D0 8D 21 D0 8D 86 02 20 44 E5 60
```

If you disassemble that, you will get the following:-

```
LDA  #$00
STA  $D020
STA  $D021
STA  $0286
JSR  $E544
RTS
```

A simple way of encrypting that would be to EOR it all with $FF. This would alter the code to look like this:-

```
: 56 FF 72 DF 2F 72 DE 2F 72 79 FD DF BB 1A 9F
```

If you were to disassemble that, you would end up with a lot of JUNK! (So it is therefore a simple form of encryption).

Encryption such as that will not be enough to protect a program from prying eyes. What you really need is a way of mixing 'password protection' and 'encryption' into one.

PASSWORD ENCRYPTION

The problem with the password-protection described above was that the pass- word actually had to be stored somewhere in memory. However complex your program may be (using hidden 6502 opcodes, code-structures, or whatever kind of advanced programming), if the password is stored in memory - IT CAN BE FOUND! The same (obviously) applies if the password is hidden on disk, in the drives RAM, or whatever! So...What on earth can we possibly do to get past storing the password in memory???

Simple...Imagine the password 'GRUMPY'. The ASCII equivalent to this is:

```
: 47 52 55 4D 50 59
```

We could use these values as ENCRYPTION CODES. We would EOR the first byte of a program with $47, 2nd byte with $52, 3rd byte with $55, etc... The 7th byte would obviously be EOR'ed by $47 (restarting the series).

All that now needs to be done is to add a short 'header' to the encrypted program which asks for a password, and uses the ASCII values of the password as DECRYPTION CODES (used in EXACTLY the same way as encryption codes, using the EOR command again!). If the correct password is entered (GRUMPY), then the program will be decrypted back to it's original form. If the wrong password was entered, then the program will be converted to JUNK, and the computer will most likely crash!

I have provided a small (and very simple) utility which can be used to encrypt BASIC programs. To password-protect a program, just follow these simple procedures:-

1. **Load the "ENCRYPTER.SYS" file (,8,1).**
2. **Load the program to be encrypted.**
3. **SYS 49152.**
4. **Enter the password you wish to use - max.16 chars.**
5. **Wait while the program is encrypted.**
6. **Save out the resulting program as a BASIC file.**

ENDING NOTE: If you Password-Encrypt a program, make sure you remember what the password is!! - If you destroy the original file and forget what the password was, you can consider your program 'lost' forever!!!

SECURE

Protection, Passwords, Encryption - all are valuable tools for the programmer. SAKIB KHOKHAR presents his version

How often have we all said, "I wish I could protect my Basic program from prying eyes" or "If only I could make things difficult for people to see my codeing?" Well now you can. In the past CDU has published various programs for doing just such a thing. However, like all magazines, new readers and programmers are coming along every day. Therefore, at the risk of going over old ground I give you my version of "SECURE"

SIMPLICITY ITSELF

To protect a program couldn't be easier. LOAD the program named "SECURE" from the CDU disk, or copy it onto your utilities disk first. Next LOAD your BASIC program and type SYS49152. Your program will now be protected. Should you want to LIST your program, the message;

10 SYS2090:PROTECTED BY SAKIB KHOKHAR
is displayed.

Before you try to protect your Basic program, make sure it has the line number zero (0). I suggest you use the following lines or something similar as the first two lines of your program;

0 REM
1 POKE774,226:POKE775,252

The second line, line number 1 is NOT essential, but will give some added protection. The two POKEs in line number 1 will perform a system reset is someone should use the LIST command. Once you have protected your program with SYS49152 you can SAVE the program in the normal manner. Once you re-load and RUN your now protected Basic program, the RUN/STOP key is disabled and the RESTORE key will perform a system reset.

That, as they say, is all there is to it.....

KA43-5

THE OPEN SYSTEM

KARE AANESTAD

An unusual and useful external operating system for your C64

KA43/5 opens up the operating system of your Commodore 64. It gives you comprehensive internal commands and opens up for external commands. The internal commands are as you would expect, readily executable. The external commands are automatically loaded from the disk drive before they are executed.

ALL ABOUT KA43/5

KA43/5 is a relocatable machine code (MC) utility. It will highly enhance the cooperation between your computer and any Epson (or IBM) compatible printer. The printer can either be connected to the CBM Serial port or to the User port. In the latter case the User port will act as a Standard Parallel Centronics port and the only hard-ware required is a simple straight through cable. Any character, even user defined characters can be printed with high speed. Text and high resolution (Hires) graphics can be intermixed in the same document. KA43/5 goes further and is highly valuable even without a printer as it eases and extends the use of the disk drive. New external commands can easily be added at any time by the user himself.
KA43/5 can safely be used as an utility for other programs, even other utilities. KA43 wedges itself into the other programs and should therefore, if possible, be the last one to be initiated. All KA43/5 commands can be used both in direct and program mode except after an IF THEN command.

Load and run KA43/5. Choose between Serial and Centronics printer and locate the MC-utility preferably at top of Basic RAM! If you choose this location, the program will lower the necessary pointers to protect itself. If in doubt go for the default values. Turn on any printer. You can now use OPEN 4,4 and PRINT#4 or CMD 4:LIST as normal.

COMMANDS FOR PRINTER

CHARACTER TYPE FOR THE PRINTER (key in and press <RETURN>):

(LEFT ARROW)C0 CBM or your own defined characters printed exactly asused on the screen. The control codes you can use are CHR$(18) and CHR$(146) for RVS ON/OFF, CHR$(14) and CHR$(15) for Expand ON/OFF, CHR$(21) or CTRL U for toggling Underline ON/OFF, CHR$(8) for reduced linespacing (24/216") and CHR$(7) or CTRL G for Hires (parameters as preset by the (LEFT ARROW)Gm,h,l command)) intermixed with your text.
(LEFT ARROW)C0,x As above, except printer bit map density changed to x (default 76) for both text and Hires. Consult your printer manual. Try 76+128 if you have a Serial printer set for PET ASCII and not Standard ASCII (if so also use (LEFT ARROW)C3 when not in (LEFT ARROW)C0 mode)).
(LEFT ARROW)C1 Standard printer characters, upper/lower set as on screen (default). You can use your printers control codes.
(LEFT ARROW)C2 Standard printer characters with forced ASCII conversion. To be used when you will ensure character conversion from PET ASCII to Standard ASCII.
(LEFT ARROW)C3 Standard printer characters, emulation mode i.e. no character conversion.

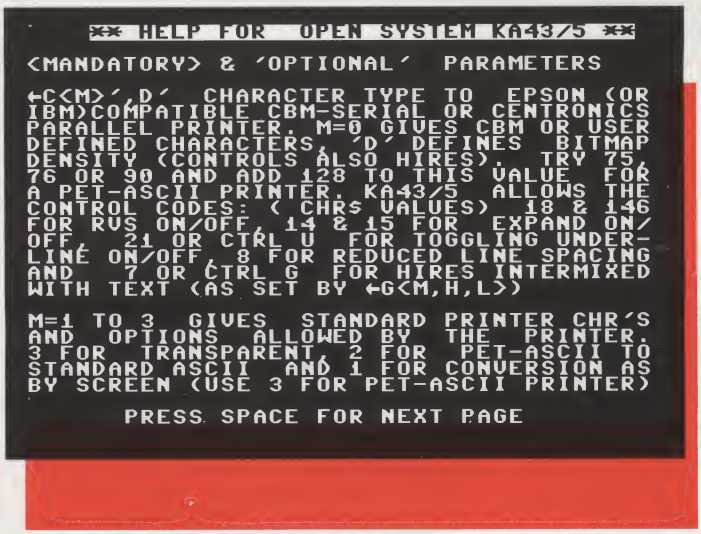
The value chosen will remain intact until you alter it or switch the computer off. Secondary address of the OPEN statement for device 4, the printer, have no effect. Listing of a program may require that you use (LEFT ARROW)C0. All Commodore control codes will then be printed.

The Character type mode will be disabled by warmstarting the computer (e.g hit RUN/STOP and RESTORE). Usually this will not disable the wedge. It is therefore easier to press: (LEFT ARROW)C1<RETURN> than to make a SYS call to restart.

SCREEN DUMPS

(LEFT ARROW)G Graphic dump of high-resolution screen. Can also be used for Multi-color, but you loose the colours. Keep RUN/STOP pressed to stop printing.
(LEFT ARROW)Gm As above with left margin set to m. Default is 0.
(LEFT ARROW)Gm,h As above, bit-map screen from location 256*h (i.e.h=high byte). Any value can be used. Most used values are 32,64,96 (default),160 and 224.
(LEFT ARROW)Gm,h,lL Gives number of graphic lines to be printed. Default is 25. This command sets the

parameters and returns without printing.
(LEFT ARROW)K Copy of text screen. You will get an exact copy as KA43/5 automatically seeks for the screen location and chr. set. Keep RUN /STOP pressed to stop printing.
(LEFT ARROW)Km As above with left margin set to m. Default is 0.
(LEFT ARROW)Km,1 As above with the one to denote reduced linespacing (24/216").
(LEFT ARROW)B0"T" Banner dump to screen of Text enclosed in quotation marks or string variable.
(LEFT ARROW)B1"T" As above to printer.
(LEFT ARROW)I Sets interrupt controlled screen dumps ON (does not work for Serial printers). KA43/5 automatically seeks the screen, it be text or Hires. Use CTRL K for printer copy. Disable with RUN/STOP RESTORE. The command will normally not work on protected programs or programs which disables the RUN/STOP and/or RESTORE keys.



DISK COMMANDS

(LEFT ARROW)D Directory is shown on the screen without erasing any Basic program. STOP the listing with the Space key.
(LEFT ARROW)E Any disk Error status is shown, i.e. disk status channel.
(LEFT ARROW)Ed Change the disk device numbers of KA43/5 commands. d to be a figure from 8 to 11.
(LEFT ARROW)E"C" Disk command, see your disk-manual. Neither OPEN nor CLOSE are required. E.g. "C"="R0:BETTER=GOOD" which renames the program from "GOOD" to "BETTER".

MC-SAVE & ADD COMMANDS

(LEFT ARROW)M<Startadr.>,<End adr.+1>,"Name",<d>,1 MC-save , d=1 for cassette and d=8 for disk. Load with LOAD"Name",d,1.
Remember to reset the Basic pointers after a MC load with NEW<RETURN>.
(LEFT ARROW)A<Adr> Additional command. <Adr.> is the SYS call address and must be a figure. Later SYS calls to the same address are made with (LEFT ARROW)A<RETURN> only. E.g. (LEFT ARROW)A49152 <RETURN>.
(LEFT ARROW)A Automatically seeks for an installed machine code monitor. List from the monitor to the printer with : OPEN 4,4:CMD 4:(LEFT ARROW)A <RETURN>

EXTERNAL COMMANDS

(LEFT ARROW)X"NAME" External command which will load a machine code program and jump to the first address loaded. The command will substitute LOAD"NAME",8,1 and the SYS number, with no Basic pointers altered. Thus the command will have the same effect as if already in memory. To recall an already loaded and executed command just press (LEFT ARROW)X <RETURN>.
Parameters can be passed. Eg. (LEFT ARROW)X"AUTO",10,10 <RETURN>, or (LEFT ARROW)X,10,10 <RETURN>. KA43/5 will protect itself from being overwritten by an External Command with an OUT OF MEMORY error and it will deny to load a Basic program starting at normal 2049.

LOCATIONS FOR KA43/5

The best location of KA43/5 OPEN SYSTEM is usually at top of Basic RAM. This area is most often not used by any other program. A draw-back is that this may unnecessary reduce the space available for Basic programs. The length of KA43/5 is 2555 bytes. You may alternatively locate KA43/5 in ML-RAM somewhere from 49152 to 53248. If location 49152 is interfering with other programs, you can choose 50693 (=53248-2555).
By locating KA43/5 at top of Basic RAM, the program will protect itself by lowering the pointers of Basic RAM top (=PEEK(55)+256*PEEK(56)). Start address (<Adr.>) will then be equal to this new RAM top (normally 38405).
You may save the ML part by help of (LEFT ARROW)M if you want. SYS<Adr.> to start up the program when it is loaded (and the computer is reset by NEW <RETURN>). If you at the same time want the program to protect itself from Basic i.e. lower the Basic top then SYS<Adr.+7>.
Be sure your printer supplies its own linefeed after carriage return (i.e. after RETURN or CHR$(13) is received). Try to avoid using file no. 125 to 127 as they may be used by KA43/5.
When (LEFT ARROW)C0, (LEFT ARROW)G, (LEFT ARROW)K or (LEFT ARROW)B are in operation, KA43/5 uses the first 2k of the RAM under the character ROM as a temporary store. This area is hardly used by any other program.
KA43/5 will work with most programs without difficulty, but some programs may use all available RAM and then wipe it out. Let's look at some programs:

KA19/2E BASIC :Locate KA43/5 at top of Basic RAM.
D-BASE KA9 :Locate KA43/5 at top of Basic RAM.
SPEEDSCRIPT 3.x :Locate KA43/5 at 50693. Load

SpeedScript. POKE 2481,197. RUN SpeedScript
DOS WEDGE:by Commodore does not like the (LEFT ARROW). To change this to e.g. ' (=SHIFT 7), POKE<Adr.+104>,39 where <Adr.> is start-address of KA43/5. All KA43/5 commands are then to be preceeded by ' ,e.g 'G.
SIMONS BASIC:Insert the cartridge. Locate KA43/5 at top of Basic RAM.

SOME POKE's can help you to customise KA43/5 to your needs:

POKE<Adr.+1776>,x - x is number of lines (LEFT ARROW)K will dump to the printer (25 by default).
POKE<Adr.+2217>,x - Where x is ASCII value of printing character used by (LEFT ARROW)B (64 by default).
POKE<Adr.+904>,x - x is Control code used for underline (21 by default). POKE<Adr.+908>,128 - For overline instead of underline.
POKE<Adr.+1736>,x - Value of reduced linespacing is x/216" (24 by default). Disable this facility with **POKE<Adr.+1705>,96.**
POKE<Adr.+2026>,2 - CBM K instead of CTRL K for Interrupt kopi (copy) to printer.
POKE<Adr.+2033>,36 - And the printing starts with only a touch on the CTRL (or CBM) key.
POKE<Adr.+1711>,234 - Five bytes at your choice, from <Adr.+1729> to <Adr.+1733> will be sent to the printer before the (LEFT ARROW)G, (LEFT ARROW)K and (LEFT ARROW)B1 commands and when using CHR$(8) and CHR$(7) in (LEFT ARROW)C0 mode (i.e. commands reducing the line spacing). The user port at the rear left side of the CBM64 can be hooked to a Centronics printer with a standard parallel cable. The same cable can also be used for Easy-Script, VizaWrite, Final cartridge etc.. The cable can be wired as follows:

CBM user port:	A	B	C	D	E	F	H	J	K	L	M	N
Centr. DB-36 :	27	11	2	3	4	5	6	7	8	9	1	19

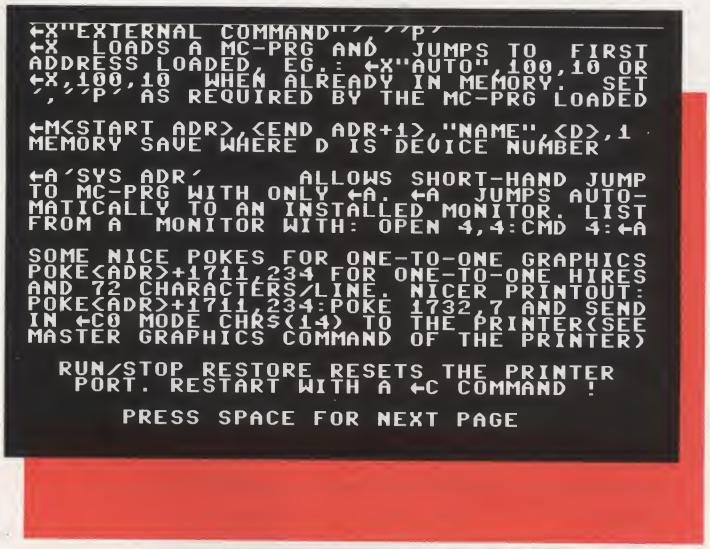
You should always connect/disconnect the printer cable at the computer end when it is not connected to the printer, and disconnect the power cord of the printer when connecting the cable to the computer. This precaution will save wear on the IC (U2, type 6526) connected to the User port.
KA43/5 OPEN SYSTEM introduces a new concept for your CBM 64. You can now let the computer grow as you expand your library of external commands. Examples of external commands are:
(LEFT ARROW)X"DELETE",100-300
(LEFT ARROW)X"RENUMBER",100,10,400-600
(Renumber the old lines 400 to 600)
(LEFT ARROW)X"OLD"

EXTERNAL COMMANDS FOR OPEN SYSTEM KA43/5

All commands have the syntax:
<left arrow>X"COMMAND"',EXPRESSION'
where <EXPRESSION> is mandatory
and 'EXPRESSION' is optional

After a command is loaded, a recall can be made with only :<left arrow>X',parameters'

Eg. : <l.arrow>X"AUTO",100,10
and later : <l.arrow>X,100,10



EXTERNAL COMMANDS

"APPEND","NAME"',D'
Where D=1 for tape and 8 for disk.
"AUTO",<line start>',step'
Auto linenumbering. Cursor down to an empty line and press <RETURN> to disable.
"BIG CHARACTERS"
Prints double height characters to the text screen. This is achieved with the same text on two adjacent lines, the second time with RVS on. Use NORSK or another foreign font to configure the computer for RAM based characters before you enter this command.
"DATA AUTO",<line start>',step'
as AUTO with DATA written automatically after the linenumber.
"DELETE",'lineno'-'lineno'
deletes all lines between given numbers.
"EASY LIST"
prints CBM control codes as readable mnemonics. Repeat <l.arrow>X to turn off EASY LIST mode.
"FILE-READER","FILE"',D",M'
Reads files, both SEQ and PRG, from disk to device D where 3 is screen (default) 4 to 7 are printers and 2 is RS232C (OPEN file 125 before sending to RS232C,eg.:OPEN 125,2,3,CHR$(8)+CHR$(1) for 1200 baud. POKE<Adr.+341>,2 if you use KA43/5 as this will let the the conversion from PET ASCII to Standard ASCII take effect on the RS232C channel. Use <l.arrow>C2). M is mode, 0 for only printable letters (all control codes off, default) while 1 gives all letters (transparent). LOCK the screen with SHIFT, and STOP with F7.
"FIND",'start lineno':<hunt>

Hunts expressions in a BASIC program. If you are hunting for reserved Basic words, they must be enclosed in single apostrophes (i.e. SHIFT 7).

"FUNCTION KEYS"
Finish key definition with left arrow for automatic 'RETURN'. RUN/STOP RESTORE for reset. Start up without redefining: POKE788,144:POKE789,192.

"HELP KA43/5"
Help menu

"KLOKKE",hour:min:sec' WATCH

"KLOKKE",B to disable (blank), but the watch keeps going. Only "KLOKKE" to turn on display again. You should disable before SAVE'ing. Both hour, min and sec must be set!

"KOPI"
Dumps text screen to printer using the character font of the printer

"MERGE","NAVN",<D>
Only for disk with D from 8 (normal disk no.) to 15. The merged file from
the disk overwrites any lines in memory with the same number.

"NORSK"
Foreign character set (others are also available on the system disk). The screen address is 52224 (=204*256), and the character set lies under the KERNAL. You can print out the character set with <l.arrow>G27,224.

"OLD"
Resets a lost BASIC program

"RENUMBER"',start",step",line-line'
Use line-line when you only want part of a Basic program (i.e. only old lines between numbers given) renumbered.

"VARIABLE DUMP"
Shows the variables and their values when a Basic program has been run.

HINT AND TIPS

It is good practice only to use one EXTERNAL COMMAND at a time. Remember that <l.arrow>X jumps to last program loaded. Keeping to this habit you will never need to remember the start address, and you avoid to overlap programs and lock-ups . NORSK and KLOKKE (i.e. foreign character sets and watch) can both be in RAM at the same time. None of these will interfere with any of the other EXTERNAL commands.

OLD, DELETE, APPEND and **MERGE** will not work on a BASIC program with a machine-code tail (eg. the KA43 loader). RENUMBER may have problems with some Basic extensions. All commands are independent of KA43/5 and can be used with their SYS<address>.

KANGAROO KORNER

The Second Segment

The second half of our useful routines from 'Down Under' gets an airing **ELAINE FOSTER.**

We started this series of useful routines with an extensive look at making your 'freezed games' behave better, overcoming some of the pitfalls encountered when using 'freeze cartridges'. On this months disk you will find the **DYN KB BOOT** program which we missed last month. Also on this months disk, along with the up and running programs we talk about in this article, are the SOURCECODE files appertaining to each of them. You can examine these sourcecode files with Your Commodore's **"SPEEDY ASSEMBLER"** or any assembler that is compatible. Alternatively, you can load the Machine Code and then examine it using a Machine Code Monitor.

We will be looking at the following routines in this months issue;

PERMANENT HEADER - An easy way of displaying a message constantly on the top of the screen.

UNIVERSAL VERIFIER - A simple method of skirting the restrictions imposed by the VERIFY command.

CUSTOM BASIC - A bit of fun that allows you to play around with Basic keywords.

The remaining 2 routines; **SPACE INSERTER** and **LOAD ADDER** will be covered in next months issue of the magazine. Until then, may I wish all of the readers in BRITAIN and the rest of the world a very happy CHRISTMAS and a prosperous NEW YEAR.

PERMANENT IRQ HEADER

This is a Machine Code program which puts a message at the top of the display screen, and it stays there even when the screen scrolls. It is nice for a scrolling database or directory, or even an ordinary Basic listing. As the list scrolls, the message flickers to draw your attention to it.

The Basic loader is on the disk as **"PERM IRQ.LDR".** Load it and list it now, to understand the following explanation.

The program includes several REM's which allow you to change the colour or message of the text, which in this case must be exactly 40 characters long (including spaces). You may use your own message in lines 70-110, but notice that it is in screen code, not ASCII. The program only takes 106 bytes at location 900 in the tape buffer. The Sourcecode may be used to place it where ever there is room for 106 bytes.

Referring to the Sourcecode, the IRQ normally scans the keyboard 60 times per second, and the vector (IRQVEC) at $0315/0315 points to the routine at $EA31 which does this. It "updates the software clock and STOP key check, blinks the cursor, maintains the tape interlock and reads the keyboard" (Mapping the Commodore 64, Compute! books). In this case the IRQVEC is changed (lines 200-260) to point to our routine at "MSG" (lines 280-370), which then continues to $EA31 as before. The result is that "MSG" runs every 1/60 sec and prints the message scanned by line 320, from the TEXT (lines 460-550). It does this by POKEing screen code characters into the screen memory (lines 330-350), and the colour into the colour memory. Line 340 ensures that a blank line follows the message line. Line 380 continues to the original IRQ vector destination, which was saved in lines 150-180.

The trouble with this is that the letters appear in lower case only. If you PRINTCHR(14) you may use upper and lower case text, but the message then contains graphics! So, if you want to print upper and lower case, use lines 400-430 instead of lines 460-550; this works because the text is now in normal ASCII characters. SYS900 starts the header, and you must SYS992 to turn it off (see lines 560-600).

IRQ rerouting is a very powerful technique, and can be used for many other applications too, eg scanning to see whether a certain key is pressed, and if so, imposing some extra command. In our case the **"PERMANENT HEADER"** can remind you how to stop a scroll, pause it, or whatever you wish. This is useful when there is a long scrolling, and you have forgotten the original instructions!

UNIVERSAL VERIFIER

(By ELAINE FOSTER and LEO GUNTHER)

The VERIFY command only works for program files, but what can you do to verify SEQUENTIAL or USER files? This problem arose recently for one of us: "Is that file on the backup disk the revised one or not?" - The answer was "UNIVERIFY/BASIC". The working part of which was;

```
180 OPEN2,8,2,"(sourcename),(filetype),R"
185 OPEN3,8,3,"(destination name),(filetype),R"
190 GET#2,A$:PRINT".";:IFSTTHEN220
200 GET#3,B$:ONSTGOTO220:IFA$=B$THEN190
210 PRINT"(down)..VERIFY
 ERROR":CLOSE2:CLOSE3:END
220 CLOSE2:CLOSE3:PRINT"(down)..OK":END
```

File 2 is opened in line 180, and file 3 in 185. A byte is taken from the source file in 190, and compared with the destination file in 200. If they are the same it goes back for another byte until either an inequality occurs or the end of the file (ST=64), and the appropriate message is printed for each.

This worked very well but was very slow. The Machine Code version in "UNIVERIFY/ML" more than doubled the speed of operation and made it unnecessary to specify a filetype. The first part of the program is in Basic, for convenience, but the speed-dependent part is in Machine Code following it (invisibly). If you only want results, simply put UNIVERIFY into your "must" utilities and use it freely; full instructions are included on the screen. If you want to know how it works and to learn something new about Machine Code, read on.

The Machine Code version turned out to be considerably more difficult than a simple translation of the Basic lines. After opening files with SA of 2 and 3 one would expect that the following routine would get

```
DRIVE.1 LDX #2 ;LFN=2
        JSR CHKIN      ;OPEN FOR INPT
        JSR GETIN      ;GET A BYTE
        STA STORE.1    ;STORE.1=2
        JSR READST     ;SOURCE EOF?
        BNE OK         ;Y:EXIT
DRIVE.2 LDX #3         ;LFN=3
        JSR CHKIN
        JSR GETIN
        STA STORE.2    ;STORE.2=3
        JSR READST
        BNE V.ERR      ;Y:EXIT
        LDA STORE.1    ;RECOVER A$
        CMP STORE.2    ;COMPARE B$
        BEQ DRIVE.1    ;CYCLE IF SAME
V.ERR   (print "Verify Error")
END     (closes files and exit)
```

the bytes and compare them;

If the two bytes are equal it recycles to get another one from the source. If they are different it falls through to the next routine which prints "VERIFY ERROR!" and closes both files in the usual way (CLOSE and CLRCHN). If the bytes are the same to the very end the BNE after the Source READST goes to an "OK!" message, and then closes etc. The destination READST is slightly different, because if the destination file ends before the source one they are obviously different and the error message is appropriate. With a rather more detailed error trapping system this worked well as long as the two files (with different name) were on the SAME disk. But when the files were on two different drives it hung up after a cycle or two, and NOTHING could be done to solve it with GETIN. It worked in Basic with 2 drives but not in Machine Code. Hours and hours were spent on this, but without avail; if you have an idea why, please let CDU know!!

PROBLEM SOLVED

The problem was solved by using a different GET operator, namely ACPTR at $FFA5. This gets a byte from the serial bus (not from the keyboard), like GETIN. But it differs from GETIN in two ways. It does not need to open a channel if no filename is involved (as when addressing the Command Channel of the disk (15) or the printer), and it can be cancelled by UNTLK at $FFAB. Because UNIVERIFY did need a source and destination filename, the UNTLK was the part needed here. Just as GETIN requires CHKIN to define an input channel, so ACPTR must be preceeded by TALK ($FFB4) and TKSA ($FF96) to prepare the serial bus to send data to the computer. TALK prepares the bus and TKSA sets the Secondary Address. LISTEN, SECOND and UNLISTEN are the sending equivalents (like CHKOUT); CIOUT is the equivalent of CHROUT. Thus, after OPENing in the usual way (to define the filenames and put them into a table) and so on Unfortunately, this did not work until the Input Buffer was cleared thus;

```
BUFFER  LDY #0
        LDA #0          ;NULL BYTE
LOOP    STA $0200,Y     ;INTO BUFFER
        INY
        CPY #3          ;CLEAR 3 BYTES
        BNE LOOP
        RTS
```

and this was inserted as JSR BUFFER just after the JSR UNTLK. This worked. It is worth remembering for this, and for cases when no file name is needed it can save much space. (See ANATOMY OF THE C64 for a directory routine using TALK). It may interest you to know that a marvellous debugging routine was found and can be inserted following the above program.

```
ASCII   PHA             ;STORE .A
        TAX             ;LOW BYTE
        LDA #0          ;HIGH BYTE
        JSR LINPRT      ;PRINTS ASCII
        PLA             ;RECOVER .A
        RTS
```

The result of this is that LINPRT ($BDCD) actually prints the ASCII (ie; ordinary number) of the two byte integer whose low byte is in "X" and high byte in "A". This is not as obvious as you may think. If .A=1, JSR CHROUT will give you nothing, because CHR$(1) prints nothing (worse, CHR$(5) gives a white character, and if your screen is white it disappears). LINPRT translates that 1 to ASCII 49 ($31) which prints as "1". This is a very powerful debugging tool: When you insert JSR ASCII anyplace in your program it prints the number which is in the Accumulator at that point, and it is "transparent". Debugging Machine Code is hard enough to prove the value of a good tool like this one.

FINALLY

The final Machine Code routine is located at 2800, just above the ending address of the Basic part. The various fixed constants are POKEd in from Basic and the main routine called by SYS2800. The result is that when you follow the prompts for number of drives and for the filenames it rapidly prints a dot for each byte compared, and gives either an OK! or VERIFY ERROR! for the normal situation. Where a disk is left out of a drive, or where the wrong filename is used a DISK ERROR! message appears. Therefore this is quite a user friendly, and relatively fast, although for a long text file it will take a while. If can be used for ANY filetype, and is quite unconcerned wither a file is PRG, SEQ or USR. The dots increase scan time negligibly, and are more cheerful than staring at a blank screen while the drives churn. They also give useful information. If only a few dots appear before an error, you probably have the wrong file.

CUSTOM BASIC

Many articles in computer magazines have described changes to the C64's primitive Basic 2.0, but most have intercepted various vectors. This one is simpler, and it can be changed easily. The method: Basic ROM is transferred to the RAM underneath, that RAM is then changed by POKEing. This changes the Basic interpreter, and you can make it do whatever you please. You can fix various bugs of Basic 2.0 or you can fashion it to your own taste (if you have a disassembly of the Basic ROM{*}). The idea is not new, but here a very efficient transfer method is used.

BULK TRANSFER

Line 50 of CUSTOM BASIC shows the simple sequential ROM transfer which is usually seen in computer articles, it takes 37 seconds, which seems forever. Lines 100-130 are much faster, using Basic's own mass-transfer routine at 41919 ($A3BF), which it uses often to move code. Without the REM's it only takes 2 lines, and even in Basic it is 37 times faster! This means that it can be put onto the front end of any ordinary Basic program where you want the advantages of CUSTOM BASIC, with very little delay in the setup time. Therefore, in customising you own Basic, do not use line 50 of CUSTOM BASIC. For the other routines here, delete all REM's if you need speed.

CHANGING RAM

Once Basic ROM has been transferred to the RAM underneath you have a very powerful tool. If you have a good disassembly map{*} and a simple idea of Machine Code you can change anything you like in Basic. After the transfer has been made, Basic ROM is turned off by POKE1,54 and then the interpreter looks at RAM which you can change easily. All of the advantages of RAM and no viruses! Furthermore, it can be done from Basic using only POKEs and perhaps a few DATA statements. These routines are only suggestions, try your own too.

APPLICATIONS

This program changes only five commands of Basic (and plays with a Word Table) but others are possible. For Machine Code enthusiasts brief REM's have been included to show HEX locations.

Line 230 removes very simply a major problem of INPUT, which crashes if you use a comma or colon in the text. With this modification you may use any text you like]\. Why should you want to have any "prohibited" characters for INPUT? But if you do you can substitute their ASCII for either or both of the zeros in line 230.

Line 310 allows you to use "+" instead of "," between the numbers in DATA statements. Then, when entering DATA from an article you can keep your fingers on the top line of keys, and between each number the little finger of your right hand reaches over and presses the "+". This is nicer than inserting the commas during typing or later, and much simpler than various Machine Code solutions. It only requires a change at one location (44183)! When you have to type several pages of numbers this feature is really appreciated. Its only disadvantage is that the new DATA lines can only be read if the RAM has been changed at that location, but for a long program it is worth it (and you have all the other advantages of CUSTOM BASIC).

Lines 410-420 change that awful "READY" prompt to one of your choice. If you replace it by your name it can remind you (and others) that this is your program. Any text is possible if it is exactly 6 characters long (including spaces). Other possibilities are "OK", "CMD?", ",", etc. Simply use the appropriate 6 ASCII numbers in the DATA of line 420. This works for the Printer, Disk Drive or the Screen: If you substitute your name for "READY" it will appear at the end of any listing sent to the printer.

Line 510 changes the "?" INPUT prompt to "!", but you could use any other single character for the value of X. Furthermore, it can be changed anywhere in the program by POKEing the appropriate ASCII into 43846 (see line 510). INPUT used anywhere else in the program will then have the new prompt unless it is changed. An added advantage is that when there is an error message, the "?" at the beginning is replaced by your new prompt.

Line 610 makes the very useful correction of NULL ASC. On your keyboard enter the following;

PRINTASC("")

the result will be;

?ILLEGAL QUANTITY ERROR
READY.

But with the modification used her you will have instead;

PRINTASC("")
0
ELAINE (in this case).

This is lovely if you are taking bytes from the disk, and you can simplify your programs by avoiding awkward (and slow) correctors like;

IFA$=""THENA$=CHR$(0)
or
B=ASC(A$+CHR$(0))

Enter the following test program at the end of a COPY of CUSTOM BASIC;

```
1000 OPEN1,8,0,"$0:" : REM POKE1,54
1010 FORN=0TO9:GET#1,A$
1020 REM IFA$=""THENA$=CHR$(0)
1030 PRINTASC(A$);:NEXT
1040 CLOSE1
```

With a disk in the drive RUN1000. Result: ?SYNTAX ERROR, and the drive's red light will remain on. Enter CLOSE1 to turn it off. Now delete the letters REM from line 1020 and RUN1000 again. Now you will see some numbers, two of which should be zeros. Now RUN100 to activate CUSTOM BASIC (use STOP at line 650 to omit the test), delete line 1020 above, and delete the letters REM from line 1000. The test will run without crashing.

To use CUSTOM BASIC is was necessary to enter POKE1,54 in line 1000. This is generally true: it must be entered somewhere in a program (or in DIRECT mode) before CUSTOM BASIC is used, but it will only need to be entered once. In the listing it is in line 130. The same thing applies for Direct Mode: Enter POKE1,54 once before entering something directly from the keyboard, and you will not need to enter it again. One exception applies however. If you are using a TOOLKIT CARTRIDGE the POKE1,54 MUST be entered EACH time used in Direct Mode. This is still an improvement over the vector-interception methods, because the cartridges often override them completely.

A LINE TRACER

In CUSTOM BASIC, line 150 is an extra bonus in case you haven't seen it before. When that line is included in the program, if you insert SYSTEM in some other line, it will print that line number on the screen when it comes to it. This can be very useful for debugging Basic programs. Location 41919 is actually part of Basic's own error-reporting routine. Both uses are shown when you RUN100 in the program. This is what you see (and I have filled in the INPUT reply);

```
ENTER A STRING CONTAINING PUNCTUATION !
HELLO COMPUTER: GOOD, OH?
IN 710=HELLO COMPUTER: GOOD, OH?

ASC("")=0
!SYNTAX ERROR IN 800
```

How nice to see INPUT used without crashing from the punctuations! You see how the deliberate error in line 800 used that SYSTEM routine too. In line 710 notice that it does not produce a carriage return; the IN 710 appears on the same line as other print, which shows clearly where that print comes from. This is useful as the BRK command in Machine Code, but do not forget to remove all SYSTEM's from your program after it is working. And of course omit lines 650- and all REM's from the version you actually want to use.

OTHER ROM CHANGES...?

Let your imagination be your guide for other modifications to Basic, but do not expect that they will always work. Would it be interesting to change the **STATEMENT TABLES** which are located from 40972 to 41373 ($A00C-$A19D)? You could then make the machine respond to your own version of Basic, example using po (NOT pO) instead of POKE, but it does not work: it insisted on consulting ROM, not RAM. On the other hand it was possible to change the ASCII text of Basic Error Messages, from 41374 to 41767 ($A19E-$A327). You could for instance, change **"SYNTAX"** to **"SILLY"** or even **"SEXY"**, but there seems little practical use for that, although in line 910 "SYNTAX" has been changed to **"WORD"** if you do not like the former. The simplest way to make any tentative changes is to use a Machine Code Monitor to change the RAM after transferring the ROM to RAM, exit the Monitor and test the result. If you like it then you can build it into your own **CUSTOM BASIC** by POKEs.

OTHER IDEAS

It seemed as though it could be useful to build the **"SYSTEM"** test into LET, otherwise useless?, to tell you the line number when LET was used. It worked, but inserting the necessary JMP $BDC2 at $A9A5 also ruined every defined variable. It seems that when you enter N=2 (or etc) it actually goes to the LET routine, which seems not to be so useless after all.

A friend suggested changing the cold-start message from ****** COMMODORE 64 BASIC V2 ****** to your own choice. That message is located from $E45F to $E4AB, and is printed from ROM when Basic is initialised at Power ON (or Reset). But although you could put your own message into the RAM underneath that, if you wanted to print it you would have to turn off the HIROM at $E000- at location 1. This could be done by POKE1,52. Try that, and will see why not; it hangs up. The reason is that when you turn off HIROM with a POKE you turn off the ability of the system to understand the POKE, and it goes crazy! You can do it in Machine Code, but when Basic is initialised it always reads from ROM anyhow, so nothing would be done. Too bad. Enter SYS58260 to show this.

So, experiment, but do not be surprised if you have difficulty. If you succeed, let **CDU** share in your knowledge.

RAM CHANGES

Try adding this to CUSTOM BASIC;

660 POKE56341,40:POKE650,128

The first POKE speeds up the cursor blink to a more useful rate (default is 64) the second one allows all keys to repeat, also useful. Within a program these show up in INPUT statements, but they are still there when you list the program, and are convenient when debugging.

USING MACHINE CODE

C.BASIC LDR on the disk does all of the above rather faster (nearly instantly) by using Machine Code. The REM's show where you can make changes. It includes the modifications 1-5 of CUSTOM BASIC. This is a Basic loader, and when it is run it will put the Machine Code at location 820, and SYS820 activates it. If for some reason the program crashes reactivate again either by SYS820 or POKE1,54 (but RESET will destroy 820)

This loader may be made part of a program, but it does occupy 471 bytes (omitting REM's). Lines 100-610 of CUSTOM BASIC require only 208 bytes when d-remmed, which is actually smaller, and it is easy merely to include (a d-remmed) CUSTOM BASIC in your program (lines 100-610). On the other hand, the actual Machine Code of the loader take only 71 bytes. If you save it to disk as Machine Code and then reload it by your main program, it will not take any room in the Basic area, and of course it will run much faster.

The Machine Code routine does not take any room from Basic, but as it stands it cannot be moved to another region. The Sourcecode is included on the disk. Listing 1 below shows how to save the Machine Code to disk, listing 2 shows how to load it into your main program.

LISTING 1

(Saving the MC to disk)

To save to disk, delete the SYS from C.BASIC LDR (line 200), and add the following lines;

```
290     PRINT"[down] SAVE  TO DISK? ":WAIT198,1:
        GETA$:IFA$<>"Y"THEN320
300     OPEN1,8,1,"BASICMOD(820)":PRINT#1,
        CHR$(52)CHR$(3);
310     FORN=820TO890:PRINT#1,CHR$(PEEK(N));:
        NEXT:CLOSE1
320     SYS820:END
```

LISTING 2

(Loading MC from Disk)

To load that saved program into an ordinary Basic one, use something like the following;

```
1 IFPEEK(820)<>169THENLOAD"BASICMOD(820)",8,1
2 SYS820: REM Rest of program follows
```

This should preceed most of the Basic program, and must definitely preceed any DIM statements.

{*} - Useful referances;

MAPPING THE COMMODORE 64 BY SHELDON LEEMON (COMPUTE BOOKS)
THE ANATOMY OF THE COMMODORE 64 BY MICHAEL ANGERHAUSEN et al (ABACUS PRESS)

That concludes this months offering. Next month we will give you the final two routines, namely,
LOAD ADDER and SPACE INSERTER..

TECHNO-INFO

A letter, a letter, my kingdom for a letter - "JASON FINCH" circa 1990

MY COMMENT

Before I start sifting through this month's mailbag I would like to wish each and every one of you a happy CHRISTMAS and NEW YEAR. If by the time you read this it has all passed, then I hope you had a good time. Secondly, I thank Mr L.J.TODD OF HENLOW, BEDFORDSHIRE for forwarding to us an original copy of the HOME ENTERTAINMENT CENTRE, free of charge, which I immediately passed on to MR WRIGHT OF SHROPSHIRE who expressed an interest in a computer version of the game of Bridge. It is very pleasing when such results are obtained.

FINDING THE 1581

Dear CDU,
I am currently looking for the Commodore 1581 three-and-a-half inch disk drive, with no success. I read in your article on Adventure Writing that you have a 1581, so I'm asking if you know where I can buy one, as nobody in Stevenage appears to have heard of it! I also required help finding a copy of the game "MISSION OMEGA", a review of which was printed a few months ago, but as yet I cannot find it anywhere. Please help me locate a copy of this game, as I liked the sound of it.
MARC BANGS, STEVENAGE.

Dear Marc,
I obtained my 1581 drive from FSSL and I am quite sure that they still stock them. There address for the purpose is FSSL, Masons Ryde, Defford Road, Pershore, Worcestershire, WR10 1AZ. The drive costs around two hundred and fifty pounds but I would telephone them first on 0386-553153 to check details. With regards to your second point, all I can do is ask if any of the readers has ever seen such a game in their local computer store, or alternatively if an owner of a computer store would like to write, I would be pleased to pass on the address of the store to Marc.

BORDER SPRITES

Dear CDU,
Could you please tell me the secret to obtaining sprites in the upper and lower borders. Also I read in an article of yours that there are only six key bytes used to get sprites in the side borders. What are they and how do I incorporate them in a program? I understand the concepts of raster-scan interrupts and have a working knowledge of assembly language.
ANDREW BROWN, HULL.

Dear Andrew,
You correctly identify that raster-scan interrupts are important. You must first set a latch at $F9 to carry out the instructions LDA #$13, STA $D011. This would usually "shrink" the screen vertically but done at the right place it will help to get rid of the border. You then need to set another latch at about $32 to carry out the instructions LDA #$1B, STA $D011. This then puts the screen back to the right length. Having carried out those simple instructions at the two specified addresses, ending of course with a JMP $EA31 or something similar, you should be able to get sprites in the upper and lower borders. Getting them in the left and right borders is far less complicated in terms of code used, but it is far more complex in actual reality because extremely precise timings are required - down to the very cycle. The six key bytes are given by the instructions DEC $D016, followed immediately by INC $D016. Done at the right instant this will open up a one pixel deep "hole" in the side borders. This must be repeated however many times you require with a sufficient time delay in between. Bear in mind that displaying the sprites will extend that delay anyway so you need to experiment with the sprites displayed where you want them. It is impossible to say exactly what the timings you require will be because it depends on a number of things. I suggest that you have a look at some of the programs that have been published in CDU to see how the programmers have got the sprites in the left and right borders there - but don't hack out any code byte for byte and use it in your own programs remember. I hope that information will at least set you on your way.

MEMORY SCANNER

Dear CDU,
Please could you help - I am unable to get the desired results with the "MEMORY SCANNER" utility. So far whenever I have tried to use it I finish up with a screen showing the "MEMORY SCANNER" program being read but not the adventure. Any help or advice you can give would be greatly appreciated.
ROBERT MARSH-HOBBS, NORTHAMPTON.

Dear Robert,
When you run the program the screen will fill with a number of symbols as the value in the top left is incremented. When the screen is full, the number will be 840. You then must press one of the function keys as shown in the top section of the display. To continue reading the memory you should press F1. The characters on the screen will then be overprinted with new ones and the value should increment from 840 to 1680, when again you must press F1 to continue. You must keep pressing F1 each time the counter stops, next at 2520, then 3360, then 4200 and so on. There may also be some misunderstanding regarding the adventure. You did not specify anything further about the adventure - there is no example adventure in the memory though when you run the program for the first time. You must find one yourself that you possess and load it first. Then you should be able to view it by pressing F1 at the appropriate times.

LETTER MAKER

Dear CDU,
I am having a problem with "LETTER MAKER". My problem is how to get the text in colour. At the moment all I am getting is black and white lettering. Can you please explain. When I press the letter "D" for Directory, I get the disk title and then some filenames all in white, but after this it tells you to press a key for the menu. But what is puzzling me is the letters are then in colour. But when I press any key for the menu I am back to the black screen and white lettering. I hope that you can help me solve this problem.
JAMES MCNALLY, KRIKINTILLOCH.

Dear James,
It is not possible to display the text for your letters in different colours due to the way that the utility has been programmed. The characters, you will have noticed, are all the same colour, and that is in fact colour number fifteen - light grey. You can if you wish alter this colour so that all the writing is displayed in, say, green or light blue or something, but it is not possible for different words or lines to be in different colours. To alter the general text colour simply load the program and type POKE10462,n before running it, where 'n' represents a number between 0 and 15 corresponding to your desired colour. Therefore if you wanted all text to be displayed in yellow then you would load the program, enter POKE10462,7 and then type RUN. I hope I have been of some assistance.

DRIVE DEVICE

Dear CDU,
When I very recently purchased my system second-hand I was lucky to be given a 1571, 1581 and MPS801 printer as well. Unfortunately neither the printer nor the two drives had any manuals. The 1571 drive (used mainly in 1541 mode) is set to device eight and I have the 1581 as device nine. Sometimes it is preferable to switch the device numbers, thus giving me the three-and-a-half inch 1581 drive device number eight and the 1571 number nine. However, this is very tedious flicking switches each time. The person who sold it all to me showed me the switches on the backs of the drives, but I now understand that there is also a software method for changing the device numbers but, as I said, I do not have the drive manuals, and besides, if the 1581 was made device eight how would I access the 1571 to make it device nine, as then there would be two device eights? Please shed some light on this subject for me.
DAREN POOLE, LEEDS.

Dear Daren,
There is indeed an alternative method for changing the device numbers of the drives and it is referred to as "softwiring" the drives. The trick is to incorporate another device number - device ten. You change the 1571 from device eight to ten, then the 1581 from nine to eight, and finally the 1571 from ten to nine. You are then left with the drives having opposite device numbers, with no errors being generated due to two drives having the same number during the proceedings. The method for changing the device number is OPEN 15,od,15, "U0>"+ CHR$(nd): CLOSE 15 where 'od' and 'nd' are the values for the old device number and new device number respectively. To show you exactly how to do what you want I have written a short program that has been included on this issue's disk, filed as TECHNO PROB. Run the program once to switch the drives around and then again to switch them back.

PET PROBLEMS

Dear CDU,
I think your magazine is better than most, catering for the more serious computer user, which is what I want to become. I think your letters page or TECHNO-INFO is excellent - please do not change your helpful attitude, as this feature alone will want me to go on buying your publication. What I want in a magazine is one which will provide me with answers to problems that I encounter when trying to use my computer for more practical uses than playing games. I have an 8050 dual drive as well as a 1541 and an 8032P printer. I have bought an IEEE

interface to enable me to use the disk drive and the printer but most programs wouldn't load with the interface connected. I have now bought a different IEEE interface and program will now load, but the print utilities that I have still won't run. In all I have spent about one hundred pounds on trying to get the printer to work with commercial software. What is the point of me mentioning this? Well, these are the sort of problems I hope can be answered by magazines such as yours.
BRIAN COLLEY, BRISTOL.

Dear Brian,
Thank you very much for all your kind comments about CDU and TECHNO-INFO. It is nice to know that the work is appreciated. From your letter I gather that you have never been able to use the printer with your print utilities and so I am wondering whether you have checked that the printer is device number four. You may have used it from BASIC as a different device number, but I don't know as you didn't supply those details. It is essential that the printer is device number four and you can check this by entering the line OPEN4,4: PRINT#4,"TEST": CLOSE 4. If this prints then there can't really be any reason that I can see for your utilities not to work, providing they don't require complicated things like the printing of graphics. It may be that the commands required by the printer are different to those given nowadays by commercial software. For example the code to enter bit image mode may be completely different - I do not hide the fact that I am not fully familiar with the old PET printers and drives, although I do know that the drives are not 100% compatible. I am therefore surprised that you can even get recent software to load on the drive due to the protection that many software houses now use that are designed to only operate with the 1541 and compatibles. I confess that I am not entirely sure what you can do further in the way of the printer because you haven't supplied me with any technical details or the programs that it won't work with. As I say, though, check the device number and the commands that it requires to see whether it is entirely compatible.

PASCAL

Dear CDU,
Please could you print in your magazine a request for a version of PASCAL on disk for the Commodore 64 because I am studying A-Level Computing and that is the language that we use. Thanks and well done on a great magazine.
NEIL MARSHALL, HEREFORD.

Dear Neil,
Your wish is my command - if anyone has a copy of such a programming language for sale, or anyone knows of where such a thing could be obtained, please send all relevant details to the TECHNO INFO headquarters - address at the end - so that I may pass them on to Neil.

PRINT DRIVERS

Dear CDU,
I read with interest how helpful you are in answering queries in CDU. I am hoping you can solve my problem. For many years I have had the excellent OCP ADVANCED ART STUDIO, yet I have never been able to use the print option. As you may know you must configure the program to your printer. In despair I have tried everything with no success. My printer is the Commodore MPS1250. It can use either seven, eight or nine pins for dot graphics. I have tried every variation I can think of, yet I cannot get the program to print the pictures. The program always locks up. With GEOPAINT I have no trouble using the MPS801 print driver. I have used the MPS801 configuration with the OCP but still no go. J have written to Commodore - as usual, no reply. Also to the distributors of ADVANCED ART STUDIO - again no reply. You are my last hope. Can you supply me with the information to configure the program correctly? I would be most grateful if you could help.
DAVID MEDLAND, CHESHIRE.

Dear David,
I can safely assure you that if you go through the configuration process simply pressing RETURN when presented with each question then the resulting file will be correctly configured to the MPS801 and you will be able to print. You must have done something wrong in the past when configuring the driver that was too small to notice. Unless your printer is faulty I can definitely say that reconfiguring the program will work - just go through every option very slowly and check and recheck before pressing RETURN that your finger hasn't slipped and hit some other key by accident. I wish you luck with the new driver. If that still doesn't work then your program may be corrupted. If that is the case then please feel free to rewrite telling me what happened.

MEMORY EXPANSION

Dear CDU,
I am very interested in a memory expansion for my 64. Do you know of such a system? And if so, could you point me in the right direction. Basically I'm after faster everything (with icons as well if possible).
JIM MORRIS, THE BRITISH FORCES.

Dear Jim,
There is a very worthwhile system available for the 64 that gives you the use of icons to operate such things as word-processors, a spreadsheet, graphics package, desktop publishing package and many other such utilities. The software system is called GEOS, short for Graphic Environment Operating System and is available in the UK exclusively from FSSL. You should write to

them at the address which I gave in my first reply saying that you wish to purchase GEOS and the 1750 Clone, a very good memory expansion that speeds things up quite a bit, or a similar RAM Expansion Unit (REU). I should think that the company will be only too pleased to forward their latest GEOS Catalogue to you.

HARDWARE HASSLES

Dear CDU,
There is a problem that I cannot solve, and that is: I have a 1541 disk drive (working great) and a 1550C printer (also working great) but one will not work at all with the other switched on at the same time. Consequently I have to keep switching off and on all the time and I cannot print a directory. Please can someone more knowledgeable than myself come to the rescue!! I own a 64 (circa 84), a 1541 (II) and a new 1550C printer, and also a POWER CARTRIDGE which is a great help in, amongst other things, renumbering.
GERRY SMITH, CHESHIRE.

Dear Gerry,
The only thing that I could suggest is that you check thoroughly the serial connections between the computer, the drive and the printer. Try first hooking the computer to the drive and then connecting the drive to the printer, and then try hooking the computer to the printer and then the printer to the drive. This will check to see whether there is some sort of daisychaining problem. Are you sure that the extra cable that you use when both are connected isn't damaged? If everything still doesn't work, try disconnecting the POWER CARTRIDGE. Then add the drive and printer in that order and check, then disconnect everything and hook them the other way around and check. Try just the drive on its own but with the two different leads. There is no reason why the two shouldn't work together unless of course there is some problem with the connections inside the printer or the drive, thus preventing a chain through either the printer or the drive to the other peripheral. All these things need to be checked out. It's not really a problem that can be pinpointed to any one thing I'm afraid.

MICRO MUD

Dear CDU,
I recently purchased a role playing adventure game "MICRO MUD". Unfortunately it was evident that someone had tried to copy the disk and it had erased itself. I was lucky enough to get my money back. The problem is that although I have made exhaustive enquiries I have not been able to trace the publishers or the producers - VIRGIN GAMES LTD and MOSAIC PUBLISHING LTD. Do you know of these companies or where I can obtain a copy of this intriguing game? Maybe one of your readers has one for sale. Can you help me please?
L.J.BRITTAIN, KENT.

Dear Mr.Brittain,
Unfortunately I do not have the address of either company and I am unaware of a company from which you could purchase the game. As you said, perhaps a reader has a copy for sale. If so, I would be only too pleased to pass on any relevant information that I receive to you. So if any of you readers do know the address of either company or have a copy of the game, could you please let me know.
EDITORS BIT!! - VIRGIN GAMES LTD can be found at: 2/4 Vernon Yard, 119 Portobello Road, London W11 2DX

ASSEMBLERS

Dear CDU,
I'd like to ask a few questions regarding assemblers. I have one at present but it does not have any functions like line numbering and labels and the such. It just allows assembling and disassembling of code. I would like to know where I can obtain a copy of an assembler that allows line numbers (ie. 10 LDA #$00, 20 STA...), supports labelling (.START LDA $xxxx, CMP $xxxx, BNE START), allows names to be defined for numbers (BMPAGE=$0400, LDA #$00, STA BMPAGE), and allows data to be entered in decimal form (LDA #27 instead of LDA #$1B). Please send me any details that are relevant for the 64.
MOHAMMED SAYED, LANCASHIRE.

Dear Mohammed,
What you have got is actually termed 'a machine language monitor'. It only allows the basic sort of entering of code that converts itself instantly to numbers I would imagine from what you have said. An assembler is something totally different whereby you enter the code just like a BASIC program but with each command on a separate line. You then issue a command and then the assembler reads the program and analyses it in usually two or three passes, whereby on the third pass the program is converted to machine code. An excellent assembler was published in CDU a little while ago, called the 6510+ Assembler. It was published in Volume 2, Number 4. To get hold of a copy of that disk I would write to Alphavite requesting a copy of a magazine beyond that available from Select Subscriptions.

TIP OF THE MONTH

With the letters section complete, we now move to this months tip. It comes to you from Mr.E.H.Baker from Worksop, Notts., and concerns the BASIC INPUT command. In his letter it is laid out like a poem, but obviously here there are layout restrictions. Anyway, take it away Mr.Baker:

Do you find that a comma or semi-colon are often "?extra ignored" by the 64? To this, input strings are very prone. The question mark too, to some, can be a bore! POKE 19,1 and it will disappear, POKE 19,0 to reset - you must restore. POKE 631,34: POKE 198,0 and quotes will appear, then input text to your heart's content, not too long (length restricted) take care! Not necessary to close with quotes at the end. For tidyness a PRINT is needed here and there. So this little routine, to you, I recommend:

```
10 POKE 19,1: POKE 631,34: POKE 198,1: INPUT
"TEXT ";A$
20 PRINT: POKE 19,0
30 PRINT CHR$(147)
40 PRINT A$
```

That little routine will indeed do as Mr.Baker says - you can enter commas or colons or whatever you like without anything being chopped off! That nicely rounds off this month's mailbag selection and so if you have anything to say, would like to have a tip published, or you have any programming/software related problem please do not hesitate to write to CDU Techno Info, 11 Cook Close, Brownsover, Rugby, Warwickshire, CV21 1NG. So until next month, enjoy yourself!

32 SPRITES ON THE C64

We present a neat utility that enables the use of 32 Sprites on the C64 written by MARTIN PIPER

Your trusty Commodore 64 was in fact the first computer to have hardware sprites, and had far superior graphics and sound than any other home computer at the time that it was launched.

SPRITE MIGHT

Sprites are the most versatile objects to move, they don't rub out any background and are generally a joy to use. For most programs the standard eight sprites are great to use but for games you may want an assortment of aliens buzzing around, filling the screen. Obviously eight sprites will not fill the screen, it is for this reason that I have written this routine.

The most versatile multiple routine for the Commodore 64. This program is not another "infamous 64 sprite extravaganza", see the "Commodore 64 programmers ref guide".

32 SPRITE INFORMATION

The program that is on this months disk, "32 SPRITES", will allow up to 32 sprites to be positioned, expanded, coloured and prioritised etc completely independently of each other. As an optional extra sunroof, you can even put these into the top and bottom borders. When using more than eight sprites the program also cures the vertical wraparound that is sometimes experienced in the past.

Although each sprite is able to be controlled individually, you must remember that there are in fact up to 4 strips of eight sprites. When using 16 sprites there is one division vertically and 2 strips, with 24 sprites there are 2 divisions and 3 strips, finally, with 32 sprites there are 3 divisions and 4 strips. With the routine running at full capacity there are 5 raster interrupts going on. It is quite amazing how the humble 64 can cope. These divisions are invisible but woe betide any sprite that decides to stray beyond the divisions as the data for the next lot will mingle with others and produce a noticeable quirk. There are two general rules that must be adhered to;

1. The divisions must not be less than 32 pixels and the lowest 'Y' position for any sprite must not be less than 30, or the divisions will flicker.

2. To position the sprites and change them you have to

```
DO YOU WANT DECIMAL OR HEX NUMBERS?(D/H)
THE LOCATIONS TO USED BY THE MULTIPLEXOR
 49152-53247   ARE USED BY THE PROGRAM
 50176-50207   ARE THE 'X' POSITIONS OF
               EACH SPRITE
 50208-50239   ARE THE 'Y' POSITIONS OF
               EACH SPRITE
 50240-50303   ARE THE COLOURS FOR EACH
               SPRITE
 50272-50303   ARE THE WHATS FOR EACH
               SPRITE
               (SIMALAR TO 2040-2047)
 50304-50335   IS THE CONTROL REGISTER

 TO UPDATE THE SPRITES SYS 50688
 THIS PLOTS THEIR NEW POSITIONS INTO
 MEMORY
 TO START THE MULTIPLEXOR SYS 49316
      PRESS SPACE KEY TO CONTINUE
```

POKE into memory locations like the video chip location;

C000-C700 (49152-50944) Are used by the routine.
C400-C41F (50176-50207) Are the 'X' positions for each sprite (0-31).
C420-C43F (50208-50239) Are the 'Y' positions for each sprite (0-31).
C460-C47F (50240-50271) Are the colours for each sprite (0-31).
C480-C49F (50272-50335) Are the sprite pointers for each sprite (0-31).
(this decides what the sprites look like).
C4A0-C4BF (50336-50367) Are the volumes for the control register (0-31).

(See explanation below.)

To start the routine off SYS49316 or JMP $C0A4 and to update the sprite display SYS50688 or JMP $C600.

CONTROL REGISTER

The control register is a mixture of MSB, SPRITE DATA PRIORITY, EXPANSION and MULTICOLOUR selection. Bit 7 in each register controls the MSB of each sprite, a 1 will position the sprite into MSB area and a 0 will place it into normal screen area. Bits 6 and 5 are expansion of each sprite in X and Y direction. So a 64 will expand the sprite in the X direction and a 32 will set the sprite in the Y direction. Bit 4 controls sprite data priority and a 16 will set the sprite behind the background. Bit 3 will set

the sprite into multicolour mode. Bits 2 to 0 are unused. Bit values can be added so that if you wanted a sprite in the MSB, behind the background and in multicolour mode you would have 128+16+8 which equals 152, similarly a sprite expanded in the X and Y directions would be 32+64 which is 96.

```
THE TOP BORDER CAN BE TURNED ON (I.E
NORMAL) BY POKEING $C3A3 OR 50083 WITH 0
POKEING A 1 WILL SWITCH IT OFF SO
SPRITES CAN BE DISPLAYED UNDER IT
PRESS SPACE

  NUMBERS CAN BE DIRECTLY POKED INTO THE
  $C000 OR 49152 ONWARDS AREA
  EG. POKE 49152,90   (XPOS)
  EG. POKE 49174,90   (YPOS)
  WOULD PRINT A SPRITE 90 DOWN AND 90
ACROSS IF $C0A0-$C0A3 ARE SET
        PRESS SPACE KEY TO CONTINUE
```

BORDER CONTROL

To turn off the top and bottom borders you must change $C370 (50083) to 1, to change the borders back to normal change the 1 to a 0. Also, to change the mess that appears at the bottom, change location $3FFF (16383) to 0, changing this 0 to any other number will make blank horizontal lines appear in the borders.

THE TECHNICAL BITS

To speed up the sprite update and animation, the routine at $C600 (50688) can be bypassed. What the routine actually does is to sort out the sprites positions and swap large amounts of data into order, from the lowest to the highest sprite number. It has to do this so that the sprite display routine can print all the sprites. After the sort does its job it stores the values for each sprite into $C000-$C0A3 (49152-49315). Quite simple so far isn't it? Now for the hard part...Locations $C000-$C09F (49152-49311) are an exact image of $C400-$C49F (50176-50335) except all the data for each sprite is in order. The sort routine also updates the divisions so $C0A0-$C0A3 (49312-49315) are the raster positions for each division and must also be in order from the lowest to the highest.

If a 0 is in one of the division pointers then the division will be missed out completely and will go to the next one.

SORTING DATA

Sort-routines are things that ALL programmers will use at some time in their career. There are various different forms of sorts, and in this article we will try to cover the more popular ones!

ROBERT TROUGHTON

There isn't much difference between sorting numerics and alpha-numerics, but I want to make this as easy to understand as I can, so it will be best to start with numbers! (which would you rather sort into order... a long list of numbers, or a long list of words???). Later, if time permits, I will provide an article on sorting Alphanumerics.

SORTING NUMBERS

I will provide you with a few algorithms, and examples of what each one will do with a table of numbers, and will leave you to program the routines into the C64 yourself - in Basic, Machine Code, or whatever language you prefer to use!

Note that Left-Elements are all the numbers to the left of the present-number being checked, and the Right-Elements are those to the right...

THE INSERTION SORT

This is definitely the simplest form of sort-algorithm there is, but is far from being the fastest!

1) Start with the 2nd number.
2) Compare it with all the Left-Elements, and place in correct position.
3) Move onto next number until the last number is reached.

As an example, we will take the following list of numbers (which will be used to demonstrate all the algorithms in this article).

50 24 36 62 12 48 9 80 71

50.24
(50 is compared with 24. 24<50, so 24 in 1st position. (of 2)
24.50.36
(36 is compared with 24 and 50. 24<36<50 so 36 in 2nd position. (of 3)
24.36.50.62
(24<36<50<62 so 62 is kept in 4th position. (of 4)
24.36.50.62.12
(12<24<36<50<62 so 12 in 1st position. (of 5)
12.24.36.50.62.48
(12<24<36<48<50<62 so 48 in 4th position. (of 6)
12.24.36.48.50.62.9
(9<12<24<36<48<50<62 so 9 in 1st position. (of 7)
9.12.24.36.48.50.62.80
(80 in 8th position. (of 8)
9.12.24.36.48.50.62.80.71
(62<71<80 so 71 in 8th position. (of 9)
9.12.24.36.48.50.62.71.80
(no more numbers, so list is sorted!)

BUBBLE SORT

This sort is very commonly used, but like all sorts, it has it's problems.

The name bubble derives from the way that each number will float towards the correct place one position at a time. If you have a list of 25 numbers and the greatest is at the front of the list, the number will have to be moved 24 times!

1) Start with the first 2 numbers, and compare them.
2) If the 1st number is greater than the second, a swap is necessary. Make a note somehow that a swap was performed.
3) Move along the list one place (ie. 2nd and 3rd numbers, 3rd and 4th etc.) until there are no more pairs to check.
4) If a swap was performed at any point in the 'pass' then repeat from stage 1. If no swap took place, the list must be in the correct order.

50.24
(24<50 so swap performed).
24 50.36
(36<50 so swap performed).
24 36 50.62
(50<62 so no swap needed).
24 36 50 62.12
(12<62 so swap performed).
24 36 50 12 62.48
(48<62 so swap performed).
24 36 50 12 48 62.9
(9<62 so swap performed).
24 36 50 12 48 9 62.80
(62<80 so no swap).
24 36 50 12 48 9 62 80.71
(71<80 so swap needed. End of list reached, 2nd pass necessary).
24.36
(24<36 so no swap).
24 36.50
(36<50 so no swap).
24 36 50.12
(12<50 so swap performed).
... And so on until the list is sorted (this will be known by the fact that no swaps will be performed in the 'final pass').

SHELL SORT

Unlike the Bubble Sort, this one can move data by greater distances, and hence will probably take less time!

1) Choose an integer to determine the number of positions between which comparisons will be made. (eg. if you choose 6, on the 1st pass, the 2nd number will be compared with the 8th).
2) Start with the 1st number, and compare it with the respective number (eg.7th if a displacement of 6 was chosen). Make a swap if necessary.
3) Move onto the next number, and continue until the end of the list is reached.
4) Divide the 'displacement' by 2, keeping it as an integer, and if it is greater than 1 then repeat from stage 2. When the displacement is less than 1, the list will be sorted!

TAKING A DISPLACEMENT OF 6

50.....9..... (9<50 so swap performed).
...24.....80... (24<80 so no swap needed).
.....36.....71 (36<71 so no swap needed).
List is now:-
9 24 36 62 12 48 50 80 71

DIVIDING DISPLACEMENT BY 2... INT(6/2) = 3

9.....62..... ; no swap.
..24.....12..... ; swap.
.....36.....48..... ; no swap.
.....62.....50..... ; swap.
.....12.....80... ; no swap.
.....48.....71 ; no swap.
List is now:-
9 12 36 50 24 48 62 80 71
Repeat for INT(3/2) = 2, INT(2/2) = 1, and then the list will be sorted because INT(1/2) < 1. (The C64 rounds DOWN (truncates)).

QUICK SORT

1) Start with 2 pointers (A,B) at either end of the list (or sublist, see later).
2) Compare the 2 numbers defined by the pointers. If a swap is necessary, then do so and increment A by 1. If no swap is necessary, decrement B by 1.
3) Repeat stage 2 until the 2 pointers coincide.

Note that at this point, all numbers in the Left-Sublist are less than the number at position of pointers, and all numbers in the Right-Sublist are greater.

4) You may now either sort each sublist, or split the sublists into further smaller sublists (which you may wish to do if you have a lot of data to be sorted).

Note that quick-sorts are not meant for sorting a list into the correct order, only for providing shorter sub-lists, which can each be sorted (using one of the previously defined sublists) in a much shorter amount of time.

I hope that this small article will shed some light onto the much hated subject of sorting numbers!...I will leave Sorting alphanumeric data till another time.

Adventure writer supremo JASON FINCH continues his tutorial for all you budding writers

Initially I must give you a humungous apology for not presenting you with the programs last month that I promised. This was due to a number of technical problems but rest assured that they are here this month for you - on the disk as AW-MODULES and AW-MODULES.MC. This month's instalment is quite short in comparison to the other ones that I have given you, and that is because the topic we are discussing this month is "PARSING". That is, I am going to attempt to give you an in-depth look at how a parser, or command analysing system, works so that in the future you will be ready to program your own, or accept the one that I shall be giving you at some later date. Suffice it to say that the subject can become very complicated and I do not wish to blind you with too many complex details all at once. Settle down with a cup of tea and this article in front of you and then continue.

ONCE UPON A TIME...

First of all let us assume that you have entered the following line and then we can see how the parser and program may analyse it:

"Open the cupboard, examine the shelves, take the large candle and put it on the table then go east".

This is about the most complicated form of sentence that you are likely to come across and before we start we need to assign a few words to the computer's vocabulary. Each one of these is then given a number. So we have nouns: CANDLE 1, CUPBOARD 4, SHELVES 12, TABLE 6, IT 127. And some verbs: EAST 3, EXAMINE 7, OPEN 8, PUT 16, TAKE 10. And to round things off, some adverbs, adjectives and/or linking words: ON 1, LARGE 3.

Hopefully you appreciate the fact that you must program the computer in such a way that it has an in-built databank of as many words as you think necessary. Do not try to cram the entirity of the Oxford dictionary into the memory as it just isn't essential. As a matter of fact, about two hundred words is all you need. That may sound a lot at the moment but in a large adventure it is nothing, believe me! Now each word is given a number as shown above, so that the computer can easily remember which word it has just read from your command.

BREAKING IT DOWN

The first thing to do with the sentence is to split it into its component parts - the separate commands. For example, the parser (that is the posh name for the bit of the program that breaks down your command and checks out how to react to what you have typed) could recognise "." and "," and "!" and "and" and "then" as being characters or words that split up the sentence naturally. The line would then break down to five commands as shown below:

1. Open the cupboard (,)
2. Examine the shelves (,)
3. Take the large candle (and)
4. Put it on the table (then)
5. Go east

Each of those separate commands is scanned for verbs, linking words and nouns in that order. Any that are found

have their number stored. Zeros represent non-recognised or non-present words.

1. OPEN the CUPBOARD - Verb: 8, Link: 0, Noun1: 4, Noun2: 0
2. EXAMINE the SHELVES - Verb: 7, Link: 0, Noun1: 12, Noun2: 0
3. TAKE the LARGE CANDLE - Verb: 10, Link: 3, Noun1: 1, Noun2: 0
4. PUT IT ON the TABLE - Verb: 16, Link: 1, Noun1: 127, Noun2: 6
5. Go EAST - Verb: 3, Link: 0, Noun1: 0, Noun2: 0

Then all occurrences of the noun 127 ("it", "them", etc.) are replaced by the last value held in Noun1 before that. The information for the commands then becomes:

1. V:8, L:0, N1:4, N2:0
2. V:7, L:0, N1:12, N2:0
3. V:10, L:3, N1:1, N2:0
4. V:16, L:1, N1:1, N2:6
5. V:3, L:0, N1:0, N2:0

These can either be done all in one go with the numbers stored in a large table, or you can chop off each command in turn, analyse it and then chop off the next bit from what is left. In my opinion the former is better and although neither are particularly simple to program, the former is probably slightly easier. Whichever method you choose you will need to keep a record of how many commands are still to be processed. The next input prompt is displayed when this reaches zero. If using the first of the two above methods you must reserve a large enough area of memory for the storage of information. A convenient maximum length of the input command line is no more than 256 characters. This gives a possible 128 one letter commands which require four bytes each - a total of 512 bytes is required for the storage table.

RESERVING MEMORY

You will also need to reserve quite a large chunk of memory for the vocabulary of the computer. This is absolutely vital. Unless the computer knows what the player is saying, it can't do anything. The information for the words can be stored as set out below which allows for a maximum of 128 different verbs, 128 link words, and 128 nouns. For each one you should give as many synonyms as possible. For example - get, take and carry would all be classed as the same verb. The words can be stored as ASCII values below 128. Immediately following the word is then a value of over 128 which represents, with 128 subtracted, the verb, link or noun number that pertains to that particular word. With our previous example, the word "table" whose noun number is six would be stored as 84, 65, 66, 76, 69, 134.

Of course that method only refers to one method of storing the words and you can either choose to create your own method and to write your own parser, whether it be in BASIC or machine language, or to use mine replacing the vocabulary with whatever you wish. I doubt whether you will need a parser that accepts more complex commands unless you want the ALL, EXCEPT, EVERYTHING options that allow for commands like "Take everything except the sword and put all on the table." You could, in that situation, have "everything" and "all" defined as nouns and "except" as a link word, so long as another link, such as an adjective, is not needed.

A LESSON IN GRAMMAR

Now to the different types of verb. Transitive verbs require an object or noun. For example: CATCH the ball, OPEN the door, KILL the guard, are all examples of transitive verbs in use. LOOK, HELP, EAST are all non-transitive. It is usually best to group the transitive verbs together because then you only need a simple check in the program to see whether a noun should have been typed. For example: IF V<10 THEN.... Various other checks need to be considered when writing the parser - what happens, for instance, if the command "Light the light" is given. As it stands, the parser would simply assign the second "light" as a verb because they are checked first. You must ensure that if a verb is read and one has already been assigned, then check the nouns instead. In this way, the number for the verb "light" would be stored and when "light" was encountered again the computer would have to think 'well I've already had a verb so perhaps it is a noun'. Of course your humble 64 hasn't got a brain as such (ahh!) so you have to program it to respond in such a manner. And don't forget you must assign the noun as Noun2 if one has already been assigned. If three or more are found then simply ignore all but the first two - why not?

There may be some nonsense inputs like "Get drop the lamp" that some strange person may enter to see what happens. Using the previous method, unless "drop" was defined as a noun, although I can't see why you should ever do that, the parser would ignore it and the resultant command would be interpreted as "get lamp". Although my parser would allow this, you could include a check so that if two verbs were encountered and the second wasn't a noun either, then respond with a "Sorry, what are you going on about?" statement.

These few words have just touched upon the concept of parsing and hopefully you can see how complex your parser could become. However, you can keep it to a bare minimum and just let it accept the verb-noun input. This is quite easy in BASIC - just write a routine to split the input into separate words and then check to see what is what. Also on this issue's disk is a BASIC parser that does that. You can expand on it if you like - I have kept it very simple, nowhere near as complex as the faster machine code one.

That concludes this month's dose of information. Just remember that the computer needs a vocab list and needs to split up the command that the player enters in order that the adventure program may be able to interpret the commands into something sensible. This is undoubtedly the most complex and possibly interesting parts of the adventure programming and so if you decide to embark on a parser, I wish you the best of luck. Next time we meet will be March, so until then - pleasant parsing!!!

NUMBERS AND BYTES

Signs, complements and confusion make up this month's tutorial.

JOHN SIMPSON

With eight bits - or a byte, it is possible to represent the numbers 00000000 to 11111111, or 0 to 255. Almost immediately we will observe that there are two major obstacles to overcome here. The first being that we can only represent positive numbers, and the second that the magnitude of the number is limited to 255.

SIGNED BINARY

We use the leftmost bit of the binary number, here bit 7, to indicate whether the number is either positive or negative. If the bit is clear, ie 0 then the number is positive, and if the bit is set ie 1 then the number is negative.

eg.
```
00000001  =  +1
10000001  =  - 1
00001111  =  +15
10001111  =  - 15
01111111  =  +127
11111111  =  - 127
```

We can see that we have now reduced the magnitude of the byte to 127. The eight bits, or byte, now only register from -127, to +127.

If we require larger numbers, then we must add more bits. For example, if we use two bytes, a sixteen bit word, we increase the magnitude to 64k unsigned, or -32k to +32k signed (remember from the earlier discussion a k, or kilobyte = 1,024). If this magnitude is still too small for the requirement, then we can use three bytes or more as necessary.

If we wish to use large floating point numbers then we must use a correspondingly large amount of bytes. This is why Basic, and other languages only provide limited precision.

JUST FOR CONFUSION

Let us perform an addition in signed binary by adding together "-5" and "+7"

```
 (+7)     00000111
+(-5)   + 10000101
-------------------
 (+2)     10001100, or -12
```

Obviously the decimal result is correct but the binary result is incorrect. A true result should be: "00000010".

Clearly, binary addition of signed numbers does not appear to work. The computer must not just represent information, but must also perform arithmetic upon it.

The solution to this problem is to use a method called "two's complement". However, there is an intermediate stage called, "one's complement".

ONE'S COMPLEMENT

In one's complement, all positive numbers are represented in their 'true' binary form, but negative numbers are obtained by transforming every bit into its opposite:

eg +3 is represented by 00000011
- 3 is represented by 11111100

another example:

+2 - 00000010
- 2 - 11111101 - one's complement

As a test we shall add "-4" and "+6".

```
 (-4)     11111011  - one's complement
+(+6)   + 00000110
-------------------
=(+2)(1)  00000001
```

where (1) indicates a carry. Of course the correct result should have been "+2", or "00000010".

Another test:

```
 (-3)     11111100  - one's complement
+(-2)   + 11111101  - one's complement
-------------------
=(-5)(1)  00000001
```

Again the result is incorrect, it should have been "-5", or "11111010

We must use a further representation to 'correct' the result. This is where "two's complement" evolves.

TWO'S COMPLEMENT

In two's complement positive numbers are still represented as usual in signed binary, the same as one's

complement, however the difference in the representation of negative numbers is first to complement each bit, as we did in one's complement, but then we further add one to transform the number into two's complement.

```
eg.  +3 represented in signed binary        00000011

     - 3 (+3's one's complement)            11111100
       add 1 for two's complement           00000001

     thus two's complement of +3 is         11111101
```

Let us now try an addition:

```
  (+3)     00000011
+(+5)   + 00000101

=(+8)      00001000
```

which is correct. Remember the leftmost bit tells us the sign number

Now for subtraction (which, of course, is done by addition).

```
  (+3)     00000011
+(-5)   + 11111011 - two's complement of +5

=(+2)      11111110
```

Let us identify the result by computing the two's complement.

```
One's complement of 11111110 is    00000001
add one for two's complement       00000001

Therefore the two's complement is  00000010,
                                   or "+2"
```

Our result of 11111110 represents "-2". It is correct.

Now let us subtract "+4" and "-3" (the subtraction is performed by adding the two's complement of "+3".

```
  (+4)     00000100
+(-3)   + 11111101 - two's complement

=(+1)(1)   00000001   (1) represents the carry.
```

We can ignore the carry bit (1), and see that the result is correct. In two's complement it is possible to add or subtract signed numbers regardless of the sign using the usual rules of binary addition, the result always comes out correct, including the sign.

Two's complement is the most convenient representation to use for microprocessors. On more complex processors the one's complement may be used but it does require special circuitry to "correct the result", but this does significantly speed up the processing time.

The following examples will serve to demonstrate the rules of two's complement, however, here "C" will denote a possible carry (or borrow). This can be considered as bit eight of the result. "V" will be used to denote a two's complement overflow. This is where the sign of the result is 'accidently' changed because the numbers are to large. Essentially it is an internal carry from bit 6 to bit 7 (the sign bit).

THE CARRY

```
  (128)    10000000
+(129)  + 10000001

=(257)(1) 00000001
```

where (1) indicates a carry or "C".

The result of this demonstration requires another, extra, bit, the eighth bit. Internally the microprocessor (C64/128) uses registers which are only eight bits wide (a byte), so when storing a result only bits 0to7 are preserved. The carry, therefore, will require special attention, and is detected using special instructions. Processing the carry means either storing it somewhere, usually in another byte (the hi byte), ignoring it completely, or deciding, if the largest authorized result is 11111111 (255), that it is an error.

THE OVERFLOW

Some examples of Overflow situations during signed addition

```
bit 6.....
bit 7.....
  ..
  (+64)    01000000
+(+65)  + 01000001

=(-127)    10000001
```

An internal carry has occurred from bit 6 to 7 - this is an overflow, and, "by accident", the result is now negative. This situation must be detected so that it may be corrected.

```
  (-1)     11111111
+(-1)   + 11111111

=(-2) (1)  11111110
```

Here an overflow carry has occurred from bit 6 to 7, and a carry from bit 7 into "C" (the carry). The "C" should be ignored, and because the internal carry from bit 6 to 7 did not change the sign bit, then the result is correct.

```
  (-64)    11000000
+(-65)  +  10111111

=(+127)(1)01111111
```

In this situation there has been an internal carry from bit 6 to 7, but the result is incorrect because bit 7 has been changed, an overflow has been indicated.

The carry and the overflow indicators are called "flags", and a register known as the Status Register is put aside for the use of these, and other "flags". Each bit of the register denoting a particular flag.

When there is a carry from bit 6 to 7 the "V" flag will

become set, and the "C" flag when there is a carry from bit 7. Overflow indicates that the result of the addition of signed numbers requires more bits than are available.

MORE EXAMPLES

positive - positive		positive - positive with overflow	
00000110	(+6)	01111111	(+127)
+00001000	(+8)	+00000001	(+1)
=00001110	(+14)	=10000000	(-128)
	V=0:C=0		V=1:C=0
(CORRECT)		Invalid an overflow has occurred. (ERROR)	

positive - negative (result positive)		positive - negative (result negative)	
00000100	(+4)	00000010	(+2)
+11111110	(-2)	+11111100	(-4)
=00000010	(+2)	=11111110	(-2)
	V=0:C=1 (disregard)		V=0:C=0
(CORRECT)		(CORRECT)	

negative - negative		negative - negative (with overflow)	
11111110	(-2)	10000001	(-127)
+11111100	(-4)	+11000010	(-62)
=11111010	(-6)	=01000011	(-67)
	V=0:C=1 (disregard)		V=1:C=1
(CORRECT)		(ERROR) - An overflow has occurred,	

by adding two large negative numbers together. The result would be -189,which is too large to fit into eight bits.

BACK TO MAGNITUDE

If we want to represent larger numbers we will need to use several bytes, and in order to efficiently perform arithmetic operations it is good practice to fix the number of bytes we intend to use. This is called a fixed magnitude.

I have, so far restricted the adding of numbers to one byte because the 64/128 operates primarily on eight bits at a time. Because this restricts us to a range of -127 to +127, then clearly this is insufficient for most applications.

Next month we will continue with looking at how we can use MULTIPLE PRECISION to increase our range. Until then, keep experimenting.

DESIGNING A ROLE PLAYING GAME

Fight the good fight by GORDON HAMLETT.

Last month, I examined various ways of designing your characters for RPGs. Although you can have a lot of fun with this aspect of your program, once the character has been designed, this part of the program is largely redundant.

INTO COMBAT

There is one element of RPGs though on which the quality of of your game will stand or fall; COMBAT. Any adventurer worth his salt knows that the only way to fame and glory, not to mention the treasure, is to start off killing a few GOBLINS and then working your way up to DRAGONS and DEMONS. If combat is not for you, then stick to traditional adventures.

The scope for your own combat system is enormous. You can include as much detail as you want ranging from the fairly simplistic (BARD'S TALE), through the middle ground (ULTIMA) all the way to the advanced system of a game like DUNGEONS AND DRAGONS. It is this last system that we will be concentrating on. When you realise what can be involved, then you can make the appropriate decision as to what to leave out.

DANGEROUS ENCOUNTERS

The first thing to decide is whether an encounter must necessarily result in a battle. The answer is probably yes at the lower end of the market but as games become more sophisticated, you may wish to introduce the element of running away, talking to, trading with or simply bribing whoever is standing in front of you. Nevertheless, for the purposes of this article, we shall assume that the encounter is a hostile one.

The first decision to be made is to work out who gets the first blow in. If the battle is the result of a well

planned ambush, then there is no problem, you give the ambushing side one free hit all round and then take it from there.

Otherwise, you have to decide on first strike on either a team or individual basis. Factors affecting this might include a character's dexterity or a special item being used such as boots of speed. Traditionally, ELVES have quicker reactions than HUMANS etc. and so are less likely to be surprised. It should be perfectly possible for two opponents to strike at exactly the same time and for the results of both their blows to be taken into account.

GET IN FIRST

The concept of first strike really is important. For example, in SSI's DUNGEONS AND DRAGONS series, a magic user cannot cast a spell that particular round if he has been hit so if you can guarantee to attack first and toss a fireball into the arena, then obviously you are at a huge advantage.

If you have ever played CHESS, you will soon realise that it doesn't matter if you have some powerful pieces if they are in no position to press home their attack. The ability to manoeuvre in order to gain a tactical advantage adds considerably to any combat system.

Again, consider the three most popular commercially available games. In BARD'S TALE, you cannot move at all. The party is limited to advancing as a group. In ULTIMA, a separate combat screen is displayed and characters can either move or attack when it is their turn. In DUNGEONS AND DRAGONS, each character has a movement allowance which is determined by the weight being carried and the character's strength. The player can move round the battlefield and attack as he sees fit.

Obviously, this last system offers huge amounts of variety and scope for the player. A strong fighter can move over to protect an injured friend or a weak magic user. You can also make good use of the terrain; using a wall or tree to protect your back. You can also make intelligent decisions as to where to concentrate your attacks better (eg; attempt to reach the enemy spell casters as quickly as possible).

BATTLE EQUIPMENT

Once you have an opponent in front, the actual process of fighting one another can begin. There will be a percentage chance of successfully scoring a telling blow on your adversary. This will normally be calculated as a basic chance, modified by the armour your opponent is wearing (armour here is a general term; it is more than likely to be a thick, leathery hide on a troll or the scaly skin of a dragon.) This chance to hit might be further modified by special items or characteristics. For example, a magic sword should (unless it is cursed) give a better chance of hitting than a normal model (it will cause more damage too) Similarly, a very strong character might be able to exert a blow of such ferocity that it breaks through the enemy's shield or whatever. Remember though that the opposite should apply for very weak characters.

You do not have to be standing in front of an opponent in order to be able to hit him. Long weapons such as spears might enable you to attack over the head of a colleague. Then there are all the various missiles; arrows, crossbow bolts, boulders, slings, daggers, hand axes and so on. With these weapons, you will need to determine a maximum range together and then work out if an opponent is in line of sight on the battlefield.

Some weapons, eg flaming oil and magic spells such as fireballs might affect an area rather than an individual. Remember that everybody within that area is 'attacked', even if it includes a member of your own party.

DAMAGE POINTS

A successful hit will result in your opponent suffering damage. This is invariably measured in hit points. When someone's hit points are reduced to zero, then that character is deemed to have died. Damage is usually a random figure within a range specified for each weapon. Thus a long sword might cause 1-10 points of damage whereas a dagger might only cause 1-4. Again, these figures might be modified by magical weapons so that a '+1' long sword would cause 2-11 damage points (1-10 +1).

Again, damage can be affected by strength. Some damage depends as well on the skill of the person inflicting it. This usually applies to spells. A magic bolt might cause 1-3 points of damage per level of spell caster. This means that a novice starting out on an adventure would only cause 1-3 points. By the time he was a twelfth level magician though, the same spell would cause 12-36 points of damage.

MORE COMPLEX ROUTINES

That is more or less the main part of a combat system explained. What is written above, applies equally to the monsters attacking your party. The battle is divided up into a number of rounds or turns, with one round consisting of every person involved in the battle having one chance to do something. First strike is then determined all over again as the next round begins and so on until the battle is won, hopefully by your side.

There are a couple of odds and ends to be cleared up before moving on. Should any person turn and run away whilst the battle is in progress, any opponent that he passes directly in front of gets the chance of a free swipe. Additionally, this extra hit is aimed at the back. As well as being considerably easier to hit, all armour class bonuses due to shields and dexterity etc are no longer valid.

INVISIBILITY

Invisible creatures give themselves away as soon as they attack. You might decide that they are still difficult to hit, but at least you know where they are. CLERICS can usually call on their various gods to help them if they are in combat against evil spirits, usually called the undead; skeletons, vampires and so on. Allow one attempt per combat per cleric to see if you can dispel these evil spirits. Remember, a battle does not consist solely of moving and hitting. Characters might wish to do something else; shout a word of surrender, quaff a potion, change weapons, use a magical item and so on.

AERIAL COMBAT

Finally, aerial combat. As it is almost impossible to depict a 3D battle on a 2D monitor, my suggestion is, forget it. Remember at all times, you are the ultimate creator as far as these games go. If something doesn't fit into your story, then leave it out and use the idea elsewhere. You do not have to explain your theories to anybody else or argue with someone who wants to use a PEGASUS instead of a horse. You are the boss.

GAINING IDEAS

If you are interested in looking into the ideas behind combat or any other systems more deeply, then I would strongly recommend that you take a look at the DUNGEONMASTER'S GUIDE AND PLAYER'S manual, both published by TSR. Although they refer exclusively to the ADVANCED DUNGEONS AND DRAGONS SYSTEM, they are well thought out and go into much greater detail than I can. You won't agree with everything that they include by a long stretch but you will get a lot of ideas. Remember though, you are trying to devise your own original system, not copy somebody else's.

Once the battle is won, it is time to give out the rewards. Experience points and treasure are what every adventurer strives for and if there is one area likely to unbalance your game quicker than anything else, it is the allocation of loot.

KEEPING INTEREST ALIVE

To start with, you have got to keep the player interested. There is no point having a vicious and dangerous battle against a couple of minor demons if the only reward is a few experience points and a couple of gold pieces each. The players are not going to come back for more. But, equally importantly, do not fall into the temptation of giving too much away too quickly. If you acquire a +6 sword of giant slaying after your first encounter with a party of GOBLINS, what are you going to give them when they defeat the aforementioned giant?

FEASIBILITY

Consider too the nature of the treasure. It is alright for DRAGONS to sit on their large horde of gold because that is what dragons do. But if a GOBLIN captain has a +1 suit of armour, the chances are that he is going to wear it, not hide it away in a locked chest. A creature of very low intelligence such as a giant worm that wanders around dungeons sucking up debris is going to be full of precisely that, debris. There might be the odd worthwhile article somewhere in the slime of its gizzard but then again, there might not be.

If you decide that defeating a group of TROLLS is going to produce treasure worth 500 gold pieces, it certainly won't be in the form of a single piece of jewellery. Do you carry all your worldly goods around when you go out shopping? Of course you don't so why should you expect a raiding party of KOBOLDS to do likewise. Now, if you catch them in their lair, then that is a different matter. But now, the treasure is likely to be hidden, or trapped, or both. Intelligent monsters will know what their stuff is worth and make best possible use of it.

IN CONCLUSION

The last thing I wish to say about combat is to consider its frequency. You will have various planned encounters inside your set scenarios; castles, dungeons or whatever. It should be possible to ensure that the player tackles the scenarios in the right order so that the strength of the monsters will be about what you want it to be. But there are also random encounters as the party travels about and there is nothing worse than a group of monsters that is so strong that it wipes out your party at a stroke. Similarly, if you have a strong party, then repeated battles against monsters that you can kill without thinking, quickly leads to boredom. Ideally, a group of monsters should be about equal or up to ten per cent stronger than your party so that there is a real sense of achievement when you defeat them. Look for variety. Always prefer a few interesting scenarios to a lot of routine ones. If you have a sophisticated combat system, then fine, make full use of it. If it is more simplistic, look towards other areas of the game to keep your players coming back.

Next month I will look at that interesting aspect, MAGIC.

FURTHER ADVENTURES IN 'C'

We are nearing the end of this excellent series on programming in 'C' by JOHN SIMPSON. This month we look at ARGUMENTS and STRUCTURES

So far in this series we have covered a lot of the basic rules and procedures that go into making 'C' the excellent language that it is. These last two instalments should finish by giving you enough knowledge to go out and write some really good software.

COMMANDLINE ARGUMENTS

It would be useful if we were able to pass some parameters to the program itself from the operating system. For instance if we had a program called COPY, which copied one file into another file, or some other device, such as a printer, then we could enter a command line argument such as:

COPY FILE-1 FILE-2, or COPY FILE-1 PRINTER

This example illustrates a simple application of a technique whereby a much larger number of additional commands could be added to the operating system structure.

Well C does provide a sophisticated method of dealing with such a situation by means of two arguments which can be passed to function main() at commencement of execution.

By convention the two arguments are called 'ARGC', and 'ARGV'. Here ARGC is an integer which will contain the number of command line arguments, and ARGV is a pointer to an array of character strings which are the arguments in question.

Here are some examples of values held by argc and argv:

```
COMMAND   LINE  - "test prog 1"
argc      -       3
argv[0]   -       test
argv[1]   -       prog
argv[2]   -       1

COMMAND   LINE  - "copy file1 printer"
argc      -       3
argv[0]   -       copy
argv[1]   -       file-1
argv[2]   -       printer
```

argv[0] is the name by which the program is invoked, an so argc is always at least 1. In our examples argc was 3, so argv[1] is the first real argument, and the last is argv[argc-1] (in the examples argv[2]). Should argc equal 1, then there are no command line arguments after the program name.

Here is a small program to illustrate further:

```
/* Command lines */
#include <stdio.h>
main(argc,argv)
int argc;
char *argv[];
{
  printf("The value held in argc is %d\n",argc);
    while(—argc>0)
      printf("%s%c",*++argv,(argc>1)?' ':'\n');
}
```

The first line of the program simply prints out the value held in argc. Note, too, the method of declaring an array which is not dimensioned " [] ", this means that the array is of undetermined length.

The while loop is controlled by the statement:

while(—argc>0)

This decrements argc, and as long as it is greater than zero it will execute the body of the loop. This means that when argc is zero, the name of the program, it will terminate because there are no more arguments to print.

Let's break down the next line:

printf("%s%c",*++argv,(argc>1)?' ':'\n');

***++argv,** - deliver the contents of the address pointed to by argv+1 and increment argv to same. The reason for prefix increment is we are not interested in printing the program name.

(argc>1)?' ':'\n'); - This illustrates the subtle use of the operators (?:), they replace the if...else. In other words:- if argc is greater than 1 then a space ' ' is printed, and if not (else) then a newline '\n' is printed.

The effect is to print out the arguments of the command line, with spaces separating each argument and the whole ending with a new line.

STRUCTURES

A structure is a group of data types combined as a single

item. This allows the data to be grouped for easier manipulation under a single name.

Probably the most traditional example I could use is a 'payroll' record. We can tie together a set of attributes, such as: name, address, insurance number, salary, etc and tag this to an 'employee'. Some of these attributes could, in turn, be structures themselves, a name or an address, for example, consists of several components.

Complicated data can be more efficiently organized by using a structure, particularly in large programs, because instead of using separate entities a group of related variables can be treated as a single unit

BASICS of STRUCTURE

Let's start by using an example. A date consists of multiple parts: day/month/year - perhaps day of the year and name of the month. This will give us five variables, and they can all be grouped into a single structure:

```
struct date {
  int day;
  int month;
  int year;
  int year_day;
  char month_name[4];
};
```

The declaration of a structure uses the keyword, struct. An optional name, or tag as it is sometimes called, may follow, as within this example where I used the tag name, date. The variables mentioned in the structure are usually called structure members.

The right brace which terminated the list of the members of the structure may be followed by a list of variables:

struct { ..... } x,y,z;

is the same as:

int x,y,z;

in the sense that each statement declares x,y,z to be variables of the named type and will allocate storage space for them.

A structure declaration which is not followed by a list of variables does not allocate storage space, but will describe the shape, or template, of the structure which can then be filled later.

We could, after the terminating right brace, and in front of the semicolon, initialize the structure by defining a variable here, such as c which is a structure of type date. By omitting the variable at this point then we can define it later with a line such as:

struct date c;

We could also, if we wished, declare other variables for example:

struct date c,d,e;

which means that c,d,and e are all variables of the structure type, date.

struct date dee;

will define a variable dee which is a structure of type, date.

We can also initialize an external or static structure by following its definition with a list of initialisers:

struct date person = { 11,12,1938, "Dec" };

When we wish to refer to a member of a particular structure we use a construction of the expression in the form:

STRUCTURE-NAME.MEMBER

Here the structure member operator "." will connect the structure name and the member name. To set leap_year from structure dee of the type date, a line of code could be thus:

```
leap_year = dee.year %4    == 0
          && dee.year %100 != 0
          || dee.year %400 == 0;
```

or to check the month name:

if (strcmp (dee.year_day, "Dec") == 0) ...

We can also nest structures, and the payroll example we looked at earlier might actually look like this:

```
struct employee {
   char name[name_size];
   char address[address_size];
   char post_code[8];
   long ni_number;
   double salary;
   struct date birth_date;
   struct date hire_date;
} ;
```

In our employee structure we have two dates. If we declare:

struct employee emp;

then

emp.birth_date.month

will refer to the month of birth, as:

emp.hire_date.year

will refer to the year the employee was hired. Note that the structure member "." associates left to right.

Older versions of C did not allow for the transfer of data from one structure to another. However now if it was necessary to transfer data between two variables of the same structure type, say a and b, then the transfer is simply b=a: We can also pass structure names to functions:

record(date)

FUNCTIONS AND STRUCTURES

We can pass the component members of a structure separately, or pass a pointer to the whole thing. The first alternative can be shown in the following example:

First we need to demonstrate a small function which will set the day of the year from the month and the day. This example presumes the tables required have been declared and initialised elsewhere.

```
day_of_year(year,month,day)
int year, month, day;
{
  int i, leap;
  leap = year %4     == 0
       && year %100 != 0
       || year %400 == 0;
  for (i=1; i<month; i++) {
    day += day_tab[leap][i];
  }
  return(day);
}
```

Now we can pass individual members:

dee.yeardate = day_of_year(dee.year,dee.month,dee.day);

and the other alternative to pass a pointer, using our sub-structure hiredate above, and rewritten day_of_year, we can use:

hire_date.yearday = day_of_year(&hire_date);

The function day_of_year now requires modification because its argument is now a pointer from hire_date rather than a list of variables as was the original function earlier.

```
/* set day of year from month and day */
day_of_year(ptrdate)
struct date *ptrdate
{
  int i,day,leap;
  day  = ptrdate->day;
  leap = ptrdate->year%4   == 0 &&
       ptrdate->year%100 != 0 ||
       ptrdate->year%400 == 0;
  for(i=1; i<ptrdate->month; i++) {
    day += day_tab[leap][i];
  }
  return(day);
}
```

Let us look at this in detail. The declaration

struct date *ptrdate;

tells us that we have a structure type named date, and that a pointer " *ptrdate " points to it. The operator " -> " refers to a particular member of the structure. Therefore ptrdate->year is pointing to the year variable of the structure date. This could also have been written as: (*ptrdate).year

ARRAYS OF STRUCTURES

It is also valid to have arrays of structures, which are extremely useful for handling blocks of related data.

Let's examine a small program:

```
struct records{
    char name[20];
    int day;
    char month[3];
    int year;
  };
struct record lists[]={
    "Jenni Phillips",6,"Jul",1970,
    "Malcolm Mclean",23,"Dec",1956,
    "John Oldbuddy",11,"Dec",1821",
    "Mary Lamb",31,"Feb",1872,
    "Jack Horner"12,"Mar",1761
  };
int x,y
main()
{
  c=x=1;
  for (;;) {
    printf("Which record do you wish to view?\n");
    printf ( "(Type Q to quit ) " );
    x=getchar();
    if ( x == 'Q' ) break;
    x=x-'0';
    —x;
    if(x>=5) {
      printf("*** ERROR. VALUE INCORRECT\n");
      continue;
    }
    printf("%s\n",lists[x].name);
    printf("%d\n",lists[x].day);
    printf("%s\n",lists[x].month);
    printf("%d\n",lists[x].year);
  }
}
```

As can be seen, we have created a character array to contain the names of the persons to be held in the record, as well as their date of birth. We have declared an array of structures using the statement:

struct record lists[]

We then have the lists of names, day, month, year. The order of the data elements is important, it must match the structure member that it corresponds with. The number of array elements is automatically calculated from this list.

The function main() allows the user to access the array and print the relevant data contained onto the screen. The method of accessing each member of the list is shown within the printf() output statements. Namely:

structure-name[element number].structure-member;

It is also possible, and practical, to "nest" structures, or even arrays of structures, For example:

```
struct address{
    char street[20];
    char town[20];
    char telephone[9];
  };
struct employee{
    char name[20];
    char job[20];
    char dob[8];
  };
struct general{
    struct member now;
```

```
        struct address here;
        int workno;
        int salary;
       }
        details[3] = {
         "M. Thatch","Janitor","06/10/23","10 Down
St","London",007,52000,
         "H. Husbnd","Sampler","07/08/21","10 Down
St","London",008,28000,
         "A. Worker","Nurse","26/10/68","10 New
Street","Bedford",009,8500,
        };
    */  Rest of Program.... */
```

Accessing the information contained in the records would be similar to that outlined earlier demonstrating arrays of structures. You must consider the nature of "structured" structures. If you consider the structure array as a series of boxes (pigeon-holes), then you must, logically, go to the correct box, or array element, where inside is contained the data items. This consists of (in the above example) structure detail as well as two other boxes, namely, structure address, and structure employee. These two further boxes can then be opened to access the data which they will contain.

As you can see, structures are a most powerful programming tool which C capitalizes upon. Memory dependent, a vast network of interlinked and nested arrays and structures can be formed to give fast and accurate data retrieval.

UNIONS

Objects of different types and sizes may be held, at different times, within a variable called a UNION. These provide us with a way in which we are able to manipulate different kinds of data within a single storage area.

Suppose that some constants may be INT'S, FLOAT'S or even character pointers. The value of a particular constant must be stored within a variable of the correct type, yet it is more convenient for the management of tables if the value occupies the same amount of storage and in the same place regardless of its type. Here lies the purpose of the UNION. It will provide a single variable which is able to hold any one of several types. The syntax is based upon structures:

```
union value{
  int intval;
  float floval;
  char *ptrval;
 }unionval;
```

In this example we have a variable, UNIONVAL, which will hold the largest of the three types. Anyone of these types can be assigned to UNIONVAL and can then be used within expressions (that is, so long as this is consistent - namely that the type retrieved is the type most recently stored).

The syntax for accessing the members of the union is the same as for structures, namely:

```
    union-name.union-member
or
    union-pointer->union-member
```

Let us examine an illustration. We have a variable UNIONTYPE which we are going to use to keep track of the current type we have stored in UNIONVAL (It is up to the programmer to keep track of the current type in the union):

```
    if (uniontype == int)
      printf("%d\n", unionval.intval);
    else if (uniontype == float)
      printf("%f\n", unionval.floval);
    else if (uniontype == string)
      printf("%s\n", unionval.ptrval);
    else
      printf("*** BAD TYPE %d IN UNIONVAL\n",
uniontype);
```

Unions are allowed within structures and arrays as well as vice versa, and the notation is identical to that of nested structures.

```
    struct {
      char *name;
      int dob;
      int uniontype;
      union {
          int intval;
          float floval;
          char *ptrval;
      } unionval;
      { symboltab[NUMSYM];
```

here the variable INTVAL is referred to by:

```
    symboltab[i].unionval.intval
```

and the first character of the string PTRVAL would be:

```
    *symboltab[i].unionval.ptrval
```

TO CONCLUDE

Next issue we shall be examining INPUT and OUTPUT. These facilities are not part of the C language, and I have not emphasized them in my presentation of C thus far. However, real programs do interact with their environment in more complicated ways than shown so far, therefore, next issue we shall be looking at the "standard I/O library", a set of functions designed to provide a standard input/output system for C programs.

Until then, bye...

ADVENTURE HELP LINE

JASON FINCH begins his second series of hints and tips for all those Adventure buffs

Hello, good evening, and welcome to another edition of Adventure Helpline. We are now in stage two of the proceedings, having finished dealing with TONY ROME's epic adventure KRON last month. This month marks the dawn of a new quest - the quest to successfully inform you of how to complete another excellent adventure published by CDU in June of last year. The adventure this time round, then, is THE ASTRODUS AFFAIR written by MARK TURNER. You will have had plenty of time to play about with it and a few of you may have given up if you couldn't solve the problems. For those people I shall, hopefully, inject new life into the adventure. For others of you who are still trying to find the missing link in your adventuring, I shall give what advice I can although I won't fully reveal the step-by-step guide to completing The ASTRODUS AFFAIR for a couple of months yet.

LET US BEGIN

I approached KRON by first giving a general guide and then a much more detailed one. This time around I shall firstly set out the room descriptions and give each one a number that I shall then refer to for the rest of the time that this adventure is covered. There are twenty-four locations altogether in the adventure and I shall cover fifteen this month, and then the remaining ones together with a bit of vocabulary next month. In the first two articles there won't really be a lot of information to help you solve the thing - it will just lay down the foundations and inform you of the exits that are available to you. Those given in brackets are only available when certain conditions are satisfied, such as whether or not you have a key to a door, whilst all the other exits are always available. You may find that yet more are available but they will be discussed at a later date. So without further fuss, let us get on with the proceedings...

LOCATIONS, LOCATIONS, LOCATIONS

1

You're in a damaged, dank room, with most of the walls, and the exit south, smeared with blackness. A dishevelled generator sits in the corner.
EXIT: SOUTH 2

2

This is the far end of a grey corridor leading off to the

south. A door leads north and a murky stairwell downwards.
EXITS: NORTH 1, SOUTH 3, DOWN 17 (next month!)

3

The corridor continues along here from north to south, its metallic floor inclining.
EXITS: NORTH 2, (SOUTH 4)

4

This is the far end of the grey corridor. A large door leads east, a room lies to the south, and the corridor continues north. The door is open/closed.
EXITS: NORTH 3, SOUTH 10, (EAST 5)

5

You're standing on a tiny platform within the stabalisation chamber, projecting from the wall over a large drop to the floor below. The cylindrical metallic chamber has only two exits; the one currently to your west, and another at floor level.
EXITS: WEST 4, (DOWN 6)

6

You're at the floor of the stabalisation chamber. The floor itself consists of alternating concentric circles of gold and silver, with a small silver centre. A closed door to the west has a badly damaged alpha-numeric keypad beside it.
EXIT: (UP 5)

7

You're at the crew's mess. Various tables, stools and foods are strewn about, whilst in one corner a body quietly decomposes. The only exit is to the south.
EXIT: SOUTH 11

8

This is the corner of a lengthy corridor leading from south to east. Although it seems safe, you have a strange feeling here...
EXITS: SOUTH 12, CRAWL 9

9

This is the late Captain Gontra's luxurious quarters. Chairs, desks, filing cabinets, etc line the walls, whilst a plush carpet covers the floor. A security beam guards the only door, west.
EXITS: CRAWL 8, (ENTER 13)

10

This is the engine room of the Astrodus. Although simplistic, the drive itself is extremely powerful. A passage leads north.
EXIT: NORTH 4

11

This is the crew's area, with numerous doors leading to the tiny, functional rooms, all of which have been ruthlessly pillaged. EXITS lead north and east.
EXITS: NORTH 7, EAST 12

12

You're at a spacious lounge. Extremely comfortable glomuchairs fill the room, and doors lead off to the north, west and south. A small silver disc is set into the floor.
EXITS: NORTH 8, SOUTH 16, WEST 11

13

Although dark and gloomy, you can just make out the features of this carefully hidden room. The hole is the sole exit.
EXIT: OUT 9

14 (Start)

You're at the Bridge of the Astrodus, or what remains of it. Debris and ruined machinery betray an ugly battle, Captain Gontra being one of the many casualties. Doors lead south and east. A drawer is set into one console.
EXITS: SOUTH 18, (EAST 15)

15

You're in a plain, compact corridor leading directly west to east. A small silver disc is set into the floor.
EXITS: EAST 16, WEST 14

Well that, I'm afraid, wraps up this month's instalment. As I have already said, next month we will look at the remaining locations in this same brief format together with a bit of vocabulary. After that I shall start to provide you with some information that should help you to complete this excellent adventure. See you all again next month. Have a happy holiday!

CBM/PC-LINK

At long last, a bit of kit to enable you to hook up your C64 peripherals to PC hardware

BONES

Over the last year or so, there has been a substantial increase in the purchase of IBM and Compatible PCs - most especially with the advent of the cheaper Amstrads, Commodores, as well as the many 'unheard of before' makes, etc. This is truly great for all of us enthusiasts and small businesses, but with it has come some 'real' problems. For example, how to transfer all the accumulated data/files/programs/etc from the old faithfulls - C64/128, Plus/4, and VIC-20 - over to the newly purchased PC. Not only that, but how many of us have now got CBM peripherals, such as printers and disk drives which function mainly as dust gatherers! Or, to put things another way, let us say that you, our faithful reader of CDU, are seriously considering a new addition to your equipment - namely a (and didn't you guess the next word?) PC. Well, no longer do you have to worry about buying a new printer, whilst trying to sell your old CBM one... No longer do you need to worry about your disk collection which is all in CBM Format... All this has now been taken care of in one fair swoop! Yes, your Commodore printer will be easily re-adapted to use with the PC (and still, of course, with your CBM computer).

DONT JUST BIN THINGS

All those text files, and data files you may have, are now a problem no longer - easily transferable onto your new and ultra-expensive IBM (!?).

"Well", you might be asking,"how come? What do I need? Where can I get one?" And, most importantly, "how much do I need to fork out?"

First of all, what you need is called a CBMPC-LINK, and from where, or rather whom, you can obtain such a prize is:

YORK ELECTRONIC RESEARCH
THE PADDOCKS
JOCKEY LANE
HUNTINGTON
YORK
Y03 9NE
Telephone: York (0904) 610722

And the PRICE? A nicely affordable £34.50 which includes VAT and postage.

Judging from the address of York Electronic Research you might think this is a bit of a gamble - let me hastily assure you that one could not get any further from the truth!!!

WHO ARE YER

Who are YER (you can easily work out the abbrevs.)? Maybe many of you out there in CDULand will remember them. From 1986 up until 1989 they marketed their own Commodore 64 products, which included programming tools, Viewdata software, as well as their own RS232 interface. This new product from YER may seem unusual and interesting, but given their previous track record with C64 involvement it does logically follow on from their development efforts - especially considering their main work is PC development.

MONEY WELL SPENT

So, what will you get for your hard earned thirty-four and a half quid? Well, apart from the result of years of development skills, you will receive a CBM-PC cable which consists of a 6-pin DIN plug, suitable for CBM devices, and a 25-way D connector suitable for connecting to the PC, and each joined together with about six feet of multicore. With this comes the driver software on disk - when you order you will need to tell the guys and gals at YER whether you want a 5.25", or a 3.5" disk.

Care needs to be taken when connecting the cable between the PC and the peripheral. You must be sure to identify the correct port, most PCs have at least one parallel and one serial interface - this is not CBM serial, but RS232. You must identify the parallel port. However, it must be said, it is tricky to go wrong because the serial port is male orientated, and the 25-way D connector on your cable is also male orientated.

The driver software will allow CBM printers to be accessed as standard PC devices, like LPT1 and PRN. All DOS and BIOS printer facilities are fully supported, with the full ASCII character set being available on all printers, and Epson graphic emulation for some CBM dot matrix printers.

The file transfer utility will read data directly from off CBM disk drives. Files can be selected for viewing and transfer from the disk's directory which is listed in a

scrollable menu. The utility supports Text, Binary, and Basic files, and batch copying will run unattended.

GETTING UNDER WAY

When you insert your CBMPC disk in drive <A> and enter, at the DOS prompt, SETUP, the installation utility will run. This will tell you how many parallel ports you have, and will display the main menu. Here you can test the cable which comes with the package, test the printer and disk drive, deal with the first time installation, and set up - or remove the printer driver.

When you select FIRST-TIME INSTALLATION from the menu options you are asked for a target drive and a subdirectory. This is where CBMPC will be installed on your hard disk.

As I said earlier, the printer drivers support the full ASCII character set:

!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ{ }
'abcdefghijklmnopqrstuvwxyz
(plus the UK currency symbol: |)

The textmode of the basic CBM printers stops at that. However, other printers do have additional facilities such as bold, underlining, and graphics:

MPS and VIC dot-matrix
Brother HR10C
DPS 1101
Star NL/LC-10
Citizen 120

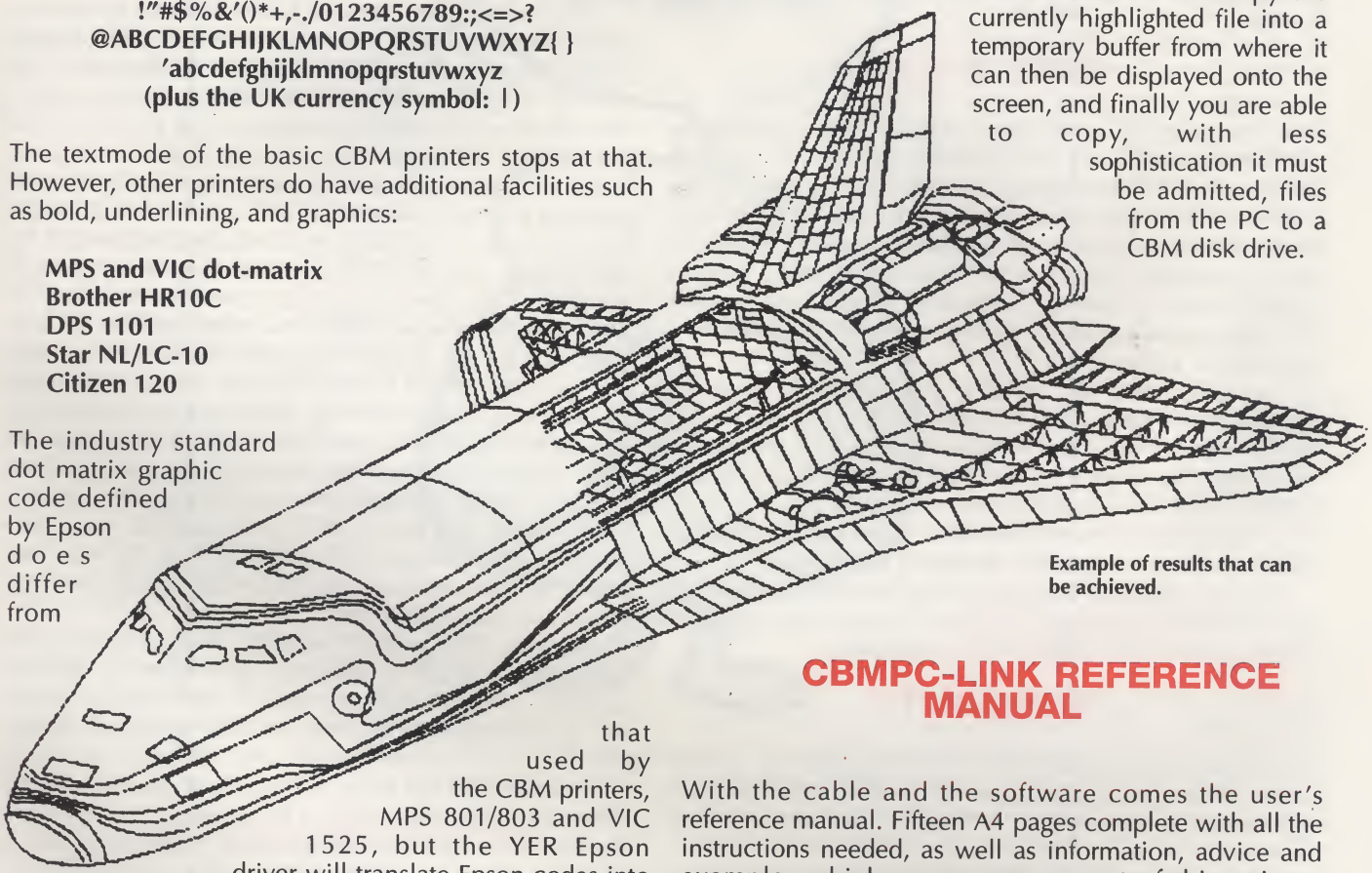
The industry standard dot matrix graphic code defined by Epson does differ from that used by the CBM printers, MPS 801/803 and VIC 1525, but the YER Epson driver will translate Epson codes into CBM graphics. Illustration.1 demonstrates a sample output of Epson graphics. This was generated using Xerox Ventura Publisher, and printed on a Commodore MPS 801 using MPSEPSON program.

CBM-2-PC FILE TRANSFER PROGRAM

Once you've got this program running, the screen shows a directory listing in the familiar CBM format, with file size in blocks, and PRG/SEQ extensions. The top line of the screen shows the disk name and ID, whilst along the bottom line is a list of 'function key' options - which does include my favourite <HELP>!

You are able to tag highlighted files - or untag them if necessary, ready for batch transfer. As you tag them, a display next to the file will state -pending=, and after a successful transfer this will change to the familiar CBM - "00,OK,00,00"=.

There are also many advanced features incorporated, such as changing file types, performing PET ASCII to real ASCII, exact copying of binary which can be displayed in hex-dump format, the detokenising of PRG files into text files suitable for PC BASICS which can then be loaded into GW-BASIC and modified to run on the PC, there is also conversion of PET screen codes to ASCII. CBM 1541 disk drives will also read old PET 2040/3040/4040 disk formats, which means PET wordcraft word processor files can easily be converted. You are also able to copy the currently highlighted file into a temporary buffer from where it can then be displayed onto the screen, and finally you are able to copy, with less sophistication it must be admitted, files from the PC to a CBM disk drive.



Example of results that can be achieved.

CBMPC-LINK REFERENCE MANUAL

With the cable and the software comes the user's reference manual. Fifteen A4 pages complete with all the instructions needed, as well as information, advice and examples which cover every aspect of this unique, unusual, interesting, and desirable package - definitely a must for those people with CBM and PC equipment at their disposal.

I could have spent another dozen or so pages celebrating this neat utility that will surely prove to be a bonus for many people, but the editor in all his wisdom, said, "hold on there bald eagle, there are other things to go in the magazine you know, Rave!"

"All right then," I softly acquiescent. "However, I must finally say, congratulations to YER for introducing the CBMPC-LINK."

PROBLEM SOLVING

We have had in the past various articles on using your mind. This is a short article by STEVEN BURGESS on the benefits of forethought and structure, with regards to programming.

If you have just had a superb idea for a computer program and have been sitting staring blank faced at a blank screen then switch off the computer and continue reading.

FIRST STEPS

The very first step in solving a problem is actually having a problem to solve. Have you had an idea for a superb program? If not then go away and do anything except rack your brains to think of a superb idea. The one sure way of clearing your mind of anything in the nature of a superb program idea is to actually try to think of one. Have a game of chess, have a sleep, a bath, watch some T.V, play some computer games, but don't try to think of an idea for a program.

Another important factor is your ability to solve the problem. Thinking out the problem will make the solving of it easier but you must be able to solve even the easiest of problems. If you are unable to program then this article is not, I'm afraid, for you.

So, once you have discovered what your problem is then you have reached the mid point of solving it.

But you must not think of solving the program in terms of commands and routines. First of all you must discover if the problem you wish to solve is feasible.

FEASIBILITY

Feasibility simply means: can the language you will be using undertake your program efficiently? Is the program you wish to write worth writing in the language you have?

If, for example, you had a sudden brainwave (whilst playing trivial pursuit) that you could write a wordprocessor, but you only knew BASIC, then would it be worth doing it? Even if you compiled the program using your 'up to 40 times faster' compiler would the program be sufficiently fast enough?

This particular problem is rather hard to gauge and would probably involve the use of compiler tests - small programs of your own creation which perform scaled down versions of functions which you wish to include. Screen updating is important with a wordprocessor, so you could write a small program to dump a section of memory, which could be the text, to the screen. If this takes ages when compiled then a wordprocessor is a definite no-no.

Another thing to consider is the need for graphics. A word-processor would not require them. A DTP package, on the other hand, would. Using unextended 64 BASIC to write a DTP package would require a great deal of staying power and patience on the parts of both programmer and user.

So, to execute a feasibility study you must look at your problem and what features it will need to be successful. Then you must look at your computer and see if it is capable of carrying out or providing these features speedily and, more importantly, without sending you insane in the process. Then you must look at yourself. Have you got the mental stamina to finish what you will start? Is your mind logical enough to figure out any equations or programming constructions which will make the program run more smoothly? If the answer the this last question is no then, although you may finish the program, it will probably end up inordinately large and complicated. Not only will it be impossible to read by anybody else, but you will probably find that you can't read it either and this is a tremendous problem if any small error shows its head.

So, then, the personal requisites for successfully and efficiently solving your problem, apart from an ability to program, are 1) A logical mind, 2) Patience, 3) Able to see dead ends and go onto another track. 4) Staying power.

Unfortunately not all of us have these qualities in great abundance. However, with a bit of work and a wish to improve, we can adopt them and make good use of them. So when you get an urge to give your keyboard a battering, think again and make a few notes. When you are tearing out your hair because of an untraceable error, save the program and have a rest and then go through it again with a fresh eye. When you find yourself getting bored because you are writing a particularly long bit of code which isn't going to do a great deal, think of what the program will be like when it is finished, watch some t.v or read a book and then attack the program again.

Basically it's all a matter of not pushing yourself too hard. The more you work that faster your level of concentration plummets to the base line.

FEATURE BOXES

Once you have thought of the program as a whole, you must think about the individual bits and bobs that make up the program.

Write a list of the features you will require. It need not be ordered into order of execution, you may always think of something essential later on - just add it to the list.

Don't think about how you are going to actually code these separate elements of the program. Just write down a word, an identifier, and proceed onto the next.

Once you have compiled a list which you think contains everything you can go onto the next stage of development. It doesn't matter if you have inadvertently omitted something. You've not committed anything to computer memory, so you can add things and remove things later if necessary.

The next stage of development is to actually order these single elements into a flow of the program. Note this is not a detailed flowchart. It is just a diagram showing the order in which features of the program are to be executed. It is a good idea to separate these by boxes - feature boxes.

If you remember something you missed earlier when compiling your feature box diagram then just stick it in in the appropriate place. If you have a menu which, say, eight other features are accessed from then have eight lines coming from the menu box going to the respective boxes.

Always remember that this diagram IS showing the flow of the program. And it is a good idea to keep this in mind. Start at the top of a large sheet of paper and work your way down. Features from the same menu should all be on the same plain.

SUB-FEATURE BOXING

Once you have reached this stage you can begin to SUB-FEATURE BOX. You must think about each of the feature boxes you have created and try to split them down further. You need not actually write this down - just think about it, although it is a good idea to make rough notes.

Then you can see if there are any parts of the feature that, when coded, could be used by other features. Such as printing text in the middle of the screen. If this is used in several features then it is a good idea to set this up as a sub-subroutine. Or if there is a lot of disk access in the program then you could have a sub-subroutine which gets the filename, or reads a specified block into the buffer or whatever.

Then you can draw another feature box diagram. This one with the sub-subroutines not connected to the main construction. Draw the boxes like before and, whenever you come across a feature box which makes use of one or other of the sub-subroutines, then write the name of that sub-subroutine in the box with the feature name, preferably in another colour.

Once you have this fully done, it will occupy a lot of paper, you can, at last, get down to the programming of your program. You can program any part of the program in any order, and it is often better to do it backwards as this gives a different perspective on the subject.

DEBUGGING

Having written your program and tested it and found errors in it then it is a good idea to debug it. Good, because if you don't then you'll never be able to use your program ever again.

Debugging is made particularly easier when you have programmed in the style mentioned above. You simply have to take listings of the routine where the error is reported then listings of any sub-subroutines called by that error and then follow the code through, thinking, in your head, what is going on. Then make appropriate corrections. Printing out has the advantage of only printing the pertinent parts. You are not confused with miles and miles of program scrolling before your eyes.

STRUCTURED PROGRAMMING

If you have always fainted at the words STRUCTURED PROGRAMMING and found the thought of actually doing it daunting, then don't. Following the method of preparation set out above, structure, in your final program, is practically automatic. You have split the program into simple parts and that, I'm afraid, is as far as structure extends on the 64.

One point is you must never jump out of a routine. Always leave it in the correct fashion (RTS or RETURN) this keeps the stack in good order and makes everyone happy.

SO GET ON WITH IT!

I'm afraid that there is little more I can say on this subject that would not involve writing your program for you. So, quite simply, get on with it.

MULTITASKING C128

MODULES,PROGRAMS and TASKS

DAVID KELSEY

We provide a breakdown of all the modules necessary to implement our multitasking concept

Last month we gave you the background needed for implementing MULTITASKING on the C128. We also provided all the "ASM" modules needed to RUN such a system. On this months disk is the program needed to "DISPLAY" these "ASM" files so that you can see what is going on. Insert the disk into the drive and type RUN"Display.bas", this will load and run Display.obj if it is not loaded. You can then view or print out each of the files as necessary. Please note., if VIEWING the files, you can only do this in 80 column mode, the PRINT mode allows for 40 or 80 columns. The "UP KEY" moves up through the code. The "DOWN KEY" moves down through the code. The "LEFT ARROW" exits the current file and the "P" key will print the file. (S key to stop).

STOR.ASM

The module named "STOR.ASM" is the first of the modules we talked about. Refer to last months text to refresh your memory.

FINAL NOTES ON LAST MONTHS TEXT

This implementation is a simple method, however very simple there are some problems that can occur with the implementation.

A) What happens when storage is freed and there is no free space left in the table even after cleanup ?

B) Memory fragmentation. Suppose that the only free storage is
$5000-$5100 and $8000-$8200. Now we want to use some storage of length
$0250. Even though there is the storage, because it isn't continuous (ie fragmented) the storage isn't available.

A SOLUTION TO (A) COULD BE AS FOLLOWS:

Instead of defining a static table, every time a new entry is to be placed in the table, find some storage and build the entry. Then using links you could chain the entry onto the other entries. A search can be done of this chained table in the usual manner. This would be the first modification to the system, but it is a re-write of the entire table logic and the logic would be complicated. The cleanup code in the above module is a simple attempt to try to deal with this problem.
(b) is more of a problem. There are again two ways around this. Blocks of storage could be moved to try to concatenate these blocks of storage. However to do this would require the moving of large blocks of memory and if any of this is code, then it would have to be relocated (ie all absolute addresses would have to be converted). This could take quite some time to do and the delay would be noticed. Another solution is to 'page out' the code.

PAGING OUT

This is a complicated idea and is based on the fact that whenever a program is running, only one instruction is ever being executed and only one memory location, apart from the instruction, is ever needed. The rest of memory is redundent. So whole blocks of memory could be placed on disk until required. The code could then be loaded in when required anywhere and control could be transfered. This could introduce programs which were fragmented all over memory, but could still run. This concept is very complicated to implement and won't be discussed any further.

PROGRAM TABLE

The other table that the system maintains is the program table. For any program that is loaded and run, certain information is kept. Some of it is for the use by the operating system and some is helpful for display so that a person can interrogate the operating system to find out information about the programs running.

INTERRUPT INFORMATION

When a program is interrupted by the operating system to run another program, all the information that the

processor uses to run a program has to be saved.When it is time to return to that program, this saved information can be brought back and the program continues as though nothing has happened.

The processor information required to be saved is as follows:-

A - Accumulator
X - Index register X
Y - Index register Y
SR - Status register
As this is put onto the stack, all that has to be done to save stack information is to move the stack and place the **STACK POINTER** in the program table.

As described earlier in this series, the special abilities of the MMU to move the location of the stack and page zero are used. So for each program its associated page 0 and page 1 areas are stored so that they can be pointed to when the program is returned to to be executed.

The final information stored in the table is the program name and where it is stored in memory (the start and end address). When a program is removed you need to know what areas of memory are freed up.
So the final layout of the table is below.

FLAG	PRG		STRT ADDR	END ADDR	ZERO RAM	ZERO PAGE	ONE RAM	ONE PAGE	PRIORITY	STACK POINTER
1	16		2	2	1	1	1	1	1	1
NUMBER OF BYTES										

(There is an extra field in the table called PRIORITY. It isn't used at present but is explained in the expansion section later in this series).

The individual routines that are used on the table are :-
SETTAB - initialises the program table
LOCNAME - locates the table entry which has the program name specified.
LOCFREE - locate a free entry in the table.

It should be noted that this table isn't maintained very centrally. Other routines actually modify this table. These just provide some basic functions on the table. Check the module named TAB.ASM at this point.

FINAL NOTES ON THE ABOVE.

Again, the table can fill up. Therefore there is a limitation of the number of programs that can be run. The number can be increased by increasing the size of the program table definitions in the assembler code. A way to make the number of programs running unlimited (limited only by the amount of storage) is to use the chaining method described for the storage table previously.

OTHER MODULES

Before going into the 2 main sections of this operating system I want to describe some of the other modules that are required.

EQUATE

This module defines the page zero layout. It is best to define the all page zero definitions at the start of the assembler code. This is because the assembler, when it comes to a label, will automatically allocate 2 memory locations if it isn't defined. (this is a problem if you define it later to be a page 0 location). Commonly used values are also defined as equates and special memory locations are also defined here. Check module named "EQUATE V1.1.ASM"

COMMON

These routines are called by many other modules and so are called common. There are no other special connections between the routines.

READNAME

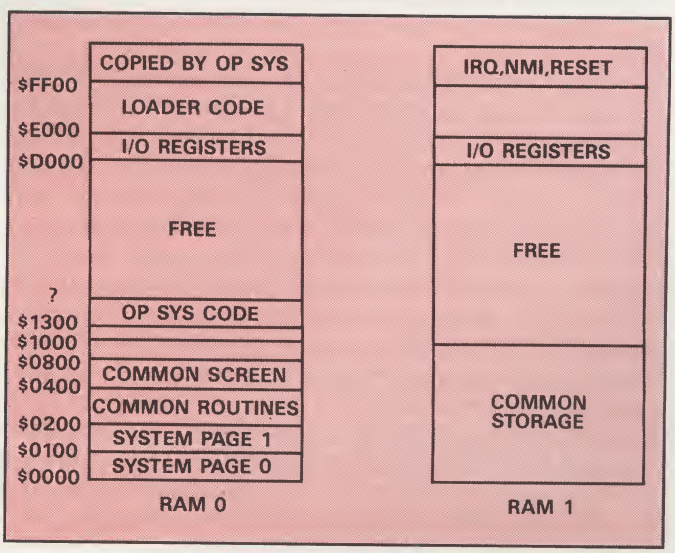
This routine is used by all the command processing modules encountered later. It will pull the program name entered on the command line off and store it in a memory block.

CONVRAM

This routine converts the BANK value which I have defined in the program table to a useable configuration that can then be stored in the configuration register. Check out module named "COMMON.ASM".

INIT

This routine sets up the 2 tables which are used by the operating system. It does other initialisation procedures which will be described later. It sets up the memory configuration that is required as well and also displays a message to say that the system is ready. The memory configuration consists of RAM, I/O + common storage. The memory map that is setup during INIT is as follows :-



With the lower 4k looking like :-

$0000 - $00FF Operating System page 0
$0100 - $01FF Operating System page 1
$0200 - $02A1 Keyboard buffer
$02A2 - $02FF LDAFAR etc routines
$0300 - $0313 Unused
$0314 - $0319 IRQ, NMI and BRK vectors
$031A - $0333 Load/save/serial vectors
$0334 - $03FF Unused
$0400 - $07FF Screen
$0800 - $0FFF Unused

(the unused areas can be used for Operating system routines which can be called by applications. See expansion section later). Check module named "INIT.ASM".

MESS
This is a centralised message routine that makes it easy to add a new message to be displayed. To add a message, enter the message and define a label to it. Then put the label in the label list at the top of the messages. Look at "MESS.ASM".

SCREEN
This routine provides a very simple screen driver for the operating system. As this operating system is to provide all control over the I/O. The operating system requires control over a screen to display messages and allow communication to the user. It relies on the screen being defined at location $0400 and works by having a cursor location pointing to the next place on the screen where the next character will go. Examine "SCREEN.ASM".

IRQ ROUTINE
This is the main routine. This routine is the one which actually swops programs around making it appear as though several programs are running. This routine has 2 main parts. The standard IRQ routine which is generated by a raster interrupt. The other is when the 8502 command BRK is encountered. this is a type of IRQ interrupt and must be catered for.

IRQ
When an interrupt occurs, The operating system is entered. The first thing that must be done is to set up the correct page zero and page 1 for the operating system. The required information about the program running must be saved. All that is required is to save the stack pointer in the program table. Pointing to the system stack when entering the operating system saves the 8502 registers as the interrupt routine has placed them on the stack. A new program must then be located. Once this has been done, then the routine must simulate a return from interrupt back to this program. This can be done by pointing to the new program's stack and pulling the stack pointer off the program table. On the return from interrupt the information is pulled off this stack and so we get the last saved information of that program going into the CPU registers. If only one program is running, then no other program has to be located. So this routine should just automatically return to the program running. The IRQ routine only checks to see if it is a raster interrupt. If it isn't then no action is taken. In he fullness of time, all the possible different IRQ's that can be generated need to be processed. This is discussed in the expansion chapter.

BRK
As an initial solution if a BRK is encountered, the program is terminated and removed from storage (ie storage is freed). All that is done is the program table entry is removed and the storage used by that program is freed. Another program is then located and it is returned to as though a normal interrupt has occurred.

PROBLEMS WHEN BRK ENCOUNTERED

There are several situations to consider here regarding what to do when a BRK is encountered and a certain number of programs are running. If only 1 program is running, and it encounters a BRK then what happens? The processor must return to some sort of code after the interrupt is completed. Therefore we return to a looping program that is encountered when the operating system is first initialised.

LOOPING PROGRAM

This logic is found in the initialisation code (INIT). It is just a loop consisting of one instruction JMP. When the system is first initialised and no programs are running, there is nothing to execute. As the CPU cannot be put into any sort of wait status and must be doing something, all that it does is loop. When programs are started and terminated by the system, if the number of programs drop from 1 to 0, the CPU must execute something, so the operating system just returns to this looping program until a new program is executed. Look at "IRQ.ASM".

THE COMMAND INTERFACE

There needs to be a way to load programs in and control their operating so some sort of command interface was devised. The need was that you should be able to request access to the operating system at any time. As NMI can never be suppressed this seemed to be the best way to access the system. So whenever the RESTORE key is pressed, the operating system prompts for a command. The NMI module does just that, it prompts for a command then decides whether the command is valid and if it is, passes the control to the appropriate module. The NMI routine Sets the interrupt flag which effectively stops all other programs from running. It could be possible to have both the operating system processing commands while all other programs were running, but at present there are some problems with that, so it isn't done.

The code is the interface that processes the input from the user. The actual routines that process the commands

are in separate modules, these will be discussed later. Note that the first cheat is in here. I use the C128 operating system to poll the keyboard.

Note that on interruption that certain information about page 0 and page 1 and the stack pointer are saved. This is because we are now in the operating system, we need to switch over to the operating system stack and page 0. It is quicker to just record this information rather than pull it off the table. Checkout "NMI.ASM".

COMMAND MODULES

Described here are the commands that can be processed and the relevant modules;

DISPLAY - To execute this command, type 'D'. The code is self explanatory. (DISP.ASM).

CLEAR - This routine just clears the operating system screen. To execute, type 'C'. It uses the clear screen feature of the screen module. (CLEAR.ASM).

PROGSTOP
The routine provides 2 commands. One is the Suspend and the other is quiesce. Suspend stops a program from executing, but leaves it in memory so that the program can be restarted up. Quiesce stops the program and removes it from memory. This memory is freed up and can be used for other purposes.

S prgname - to suspend a program
Q prgname - to quiesce a program

Some problems were encountered however on the first attempt to code this. What would happen if we interrupted a program and then tried to suspend ir quiesce it ? There is no problem in suspending or quiescing a program which isn't currently using the CPU, all that has to be done is set the flag in the table appropriately and if quiescing it then the memory must be released via the free storage routine. If we have to stop the program that is running then when the return from the NMI routine is executed, it cannot return to the program that has just been stopped. This means that you have to locate another program to return to, make it appear as though that one had been interrupted by the NMI routine and return to it. If you suspend or quiesce the final program, then we must return to the looping program which was first executed at the end of initialisation. (PROGSTOP.ASM).

RUN
This starts a program which has just been loaded or restarts a program which has been suspended. Again a problem occurs when you try to run the very first program. On return of the NMI interrupt you have to return to the program you have just specified to run. This is done by fooling the NMI routine into returning to that program by changing the information he has saved about the program he has interrupted. To use the command, type: R prgname. (RUN.ASM).

LOAD
This proved to be a real problem. In order for the operating system to find and allocate space for a program to be loaded, it has to know the length of the program. The execute address of this program must also be given to the operating system. The normal load and save routines in the C128 operating system don't use this kind of information, so some modifications need to take place.

NEW SAVE ROUTINE
This routine MUST be used to save a machine code program which is to be used in the operating system. It saves the start address, end address+1 and current execute address which you specifiy when saving the program. Giving the end address effectively defines the length of the program. Giving the execution address defines firstly where execution will start and where code begins for the relocation routine.

USING THE NEW SAVE ROUTINE

The program has to be loaded into memory first, away from the new save code, which is placed at $1300. To initiate the code, type SYS DEC("1300") in BASIC. To use, first place the execution address in location $fa and $fb. Then to save, use the machine code monitor save command as normal. For example :-

program at $3000 - $32A3, RAM 0. Execution at $3122

$FA = 22
$FB = 31

type S "program",8,03000,032A3
Refer to (SAVE.ASM).

BACK TO THE LOADER

I would have liked to design my own loading routine, but to do this properly allowing for application routines to also access the serial port would require a lot of work. I decided to cheat again and nick some of the code from the C128 operating system. To design a proper DOS for the new system will be quite a task and is part of future expansion. The C128 code had to be modified, so the whole of ROM from $e000-$ff00 is copied into memory to allow easy modification of the load kernal routine. The first modification to the loader is to read in the end address and execution address from the information on disk. This means that the length of the code can then be determined and a suitable section of storage for the placing of the information can be located. The kernal load routine resets the interrupt flag. If it does that the operating system will start swopping the programs and the system will crash. The parts of code that do this have to be removed. Finally certain kernal calls have to be removed as they serve no use and could cause problems.

The modifications are done at initialisation time and all mods are stored in a patch table. To use the load routine, type: L prgname

There is logic in the code to specify a priority. This isn't used and the priority idea will be given in the expansion section later in the article.

The loader loads the program into memory and sets up

the program table entry as well as locates areas for page 0 and page 1 for the program. Please note that Page 0 and Page 1 for a program can be anywhere in memory. They don't have to be close to the program or even in the same RAM block. ("LOAD.ASM").

RELOCATION ROUTINE

A program can be coded in any part of memory. To allow for 2 programs running that might have been coded so that they overlap in storage I decided to code a relocation routine. The first problem is how to decide what is code and what is data. A relocation routine could mistake data for code and change information that you don't want to change. The format of the program should be as follows :-

---

locations used by prg

Code

When you specify the execute address, the routine assumes that from that address to the end of the program is all program code and no data. This should be true and is discussed further in the programming restrictions section.

The main problem with this routine is after processing one assembler instruction, how many bytes do you move forward from that instruction to get to the next one. Also, it must recognise which instruction is followed by an absolute address. Once an absolute address has been found, it is then checked to see if this address is within the program block and thus needs to be changed to fit into the new address that the program has actually been placed at. The problem is still, how do you recognise the 'length' of each instruction. The first way is to just have a table of every single instruction and assign the number of bytes that that assembler instruction is. For example LDA $4000,x is a 3 byte instruction, and INX is a 1 byte instruction. There is however a relationship between the length of a machine code command and the byte value associated with that command.

1 Byte commands

ASL A	0A	LSR A	4A	RTI	40	TXA	8A
BRK	00	NOP	EA	RTS	60	TXS	9A
CLC	18	PHA	48	SEC	38	TYA	98
CLD	D8	PHP	08	SED	F8	INX	E8
CLI	58	PLA	68	SEI	78	INY	C8
CLV	B8	PLP	28	TAX	AA		
DEX	CA	ROL A	2A	TAY	A8		
DEY	88	ROR A	6A	TSX	BA		

Notice that there is a pattern here. All 1 byte commands apart
from 3 have the last 4 bits either 8 (1000) or A (1010).
If you look at the 2 byte commands and 3 byte commands in a
similar manner you will be able to see patterns for these as well.
Summarized below are the results :-

x8, AA, 00, 40, 60 - implies 1 byte

xC, xD, xE - implies 3 bytes

x9 and 1st 4 bits define an odd number, eg 59 (ie 4th bit is set) implies 3 bytes. All others are regarded as 2 byte commands.

Armed with this information the code is then as in "RELOC.ASM".

BRINGING THE MODULES TOGETHER

That's all the code needed, the only thing now is to assemble them all and put them together. Here is the module order that I used to assemble all the code. In the Lazy Grenius assembler, this file is the one that is assembled, the other bits of code are pulled in off disk to be assembled at the appropriate time.

```
ORG $1300
#INCLUDE "EQUATE.ASM"
#INCLUDE "INIT.ASM"
#INCLUDE "COMMON.ASM"
#INCLUDE "TAB.ASM"
#INCLUDE "STOR.ASM"
#INCLUDE "IRQ.ASM"
#INCLUDE "CMD1.ASM"
#INCLUDE "LOAD.ASM"
#INCLUDE "DISP.ASM"
#INCLUDE "RUN.ASM"
#INCLUDE "RELOC.ASM"
#INCLUDE "MESS.ASM"
#INCLUDE "SCREEN.ASM"
#INCLUDE "CLEAR.ASM"
```

CODING APPLICATIONS UNDER THIS OPERATING SYSTEM

Although I tried, some limitations have to be placed on coding when writing applications to run under the operating system. These are not guidelines, but rules. If these rules are not followed, you are liable to crash the computer or design a program that won't work.

RULE 1: You cannot use the configuration registers. (ie you cannot use addresses $D500-$D504 $FF00-$FF04

RULE 2: Do not try to use any of the ROM routines. They won't work.

RULE 3: Code design. To make sure that code relocation is effective there are certain ways in which code should be designed. This will be described in a minute.

RULE 4: Use the I/O registers at your own risk. Remember that several programs could access them at the same time causing chaos.

RULE 5: Using interrupts is dangerous. It may be possible to trap them yourself by changing the interrupt vectors at locations $0310-$0317 (as normal) but you MUST return to the operating system routines. You could still crash the system.

RULE 6: Be wary about using SEI and CLI. If you SEI, you stop the operating system swapping programs.
Really they shouldn't be used in application programs, only operating system routines.

RULE 7: All non-zero page locations must be contained within the code block loaded (ie a program which could be starting at $2000 - $23FE should have all its memory locations that it uses placed within that memory block). If you don't, you could affect other used storage locations which the operating system has allocated to another program.

RULE 8: Same as the I/O registers, be careful about using locations 0 and 1.

RULE 9: You cannot use coding routines such as 'LDA #<ADDRESS' or set up address tables using assembler mneumonics such as 'DW' (define word). The relocator won't cope with this.

CODE STRUCTURE

For the relocation code to be able to decipher which is code and which is just data so that it doesn't try to relocate data that it mistakes for code, there has to be some rule to follow:-
You must put all data areas at the top of the program. You can define the execute address and this is where the relocator will start relocating the code. Therefore it won't touch the data at the top of the program before the execute address.

CODING EXPANSIONS TO THE OPERATING SYSTEM

There are 2 points to consider here. One is the expansion of the code such as adding new information to the tables and the second is to provide executable routines that can be called by applications.

EXPANSIONS TO THE OPERATING SYSTEM

Possible expansions are to the program table by adding extra entries in the table, or by providing new commands to the operating system or by updating a module. There is no special programming techniques required here and just experience in assembler is required.

ROUTINES FOR APPLICATIONS

There is a hell of a lot of work here. Expansion here could be to provide facilities to control the access of I/O. Examples will be given in the expansion section later.
The most important consideration here is the concept of RE-ENTRANCY. What happens if you call a routine and during the running of this routine the program is interrupted to run another program. The next program to be run then calls the same routine. (This routine is an operating system function). Calling this routine would change some memory location which it uses. When we return to the first program that was interrupted, it would return to this common routine to find that the values that it had set have now been changed. Unexpected results would occur. It isn't possible to provide copies of these routines to all applications that may call them so how can we get around this problem What you need to do is for each different application that calls an operating system routine, have a different block of work locations.
The way larger systems do this is by getting a block of free storage on entry to the routine. This free storage stays unique to the register is only ever used to point to this block. The block then gets used to store information while the routine is processed. An interrupt now occurs. The routine is temporarily interrupted. Another part of software then calls the same routine. On entry a block of free storage is got. It is different to the first block got by the 1st caller to this routine because the area has been flagged as used. The register now points to these memory locations and so the routine is working with a different piece of storage and doesn't corrupt the 1st callers storage. As registers are saved at interruption time, the 1st caller hasn't lost its register that contains the pointer to it's block of storage. When the routine finally completes, it frees up the storage that it has used.
What this looks like is that one routine has blocks of storage allocated to it when called. It could have several blocks defined to it, each unique to the caller of the routine. This is basically simulating an individual copy of the routine to each caller. Only this way a share copy of fixed information is kept and a unique block for all changing memory is assigned to each caller. This method also copes with the fact that you don't know how many calls are going to be make to the routine that will run concurrently.

How can this concept be used on a C128? The problem is that you cannot use registers to point to memory locations. (Index registers are of no use as a register must point to the whole address for it to reference). There are no 8502 commands that provide something like:-

LD R10,(R11) - load Register 10 with the contents of the address pointed to by register 11.

The way round this is to remember the PAGE 0. Because of the facility to move PAGE 0, each application has its own PAGE 0 (and PAGE 1). So this provides the separate block of memory. When entry is made into a common routine for one application, the address $40 for that application will be different to the contents of $40 for

another application as they are effectively 2 different locations. This means that a common routine can absolutely reference a location, but it will be different for each application.
If you don't have enough storage in page 0, you could use the GET STORE routine to get a block of storage, then indirectly address this block using page0 pointers. Make sure you select the RAM block that the operating system is in. You could use any RAM block but then you would have to address the memory locations within the routine via LDAFAR and STAFAR, described in the MMU article, which could unnecessarily complicate things. Usually though, Page 0 and Page 1 would provide sufficient storage for all purposes. For any operating system routine, you will need to document which page0 locations are used so as to not conflict with other operating system routines as well as the calling application.

To code these routines requires a special way. The call will be in common memory and the code in common memory has to have the following code

1. Store the current RAM configuration (in page 0)
2. Set the RAM configuration to operating system one
3. JSR to code
4. Reset RAM configuration to the saved value
5. RTS

OTHER PROGRAMMING TECHNIQUES

SEI and CLI can be used to temporarily stop the operating system switching programs. This could be used to produce code that actually completes before another program is switched to. This means some code doesn't have to be re-entrant. Be careful using this technique. You could stop the whole system or slow other programs down significantly.

EXAMPLES

I have done a couple of examples to show the use of this system. The first is an operating system routine which converts a hexadecimal value into 2 screen characters. Also provided is the code in common storage to call the routine.
The 2 applications which I have provided are very similar. They poll the two TOO clocks and display them on the screen.

RE-ENTRANT CODE

This is the code that converts the hex number to screen characters.

* Here is the example definition of an operating system routine which can be called by any application and is thus re-entrant this code must be loaded before the operating system is initialised. Here is the conversion code.

```
VAL1    =$50
VAL2    =VAL1+1
        ORG $2F00 ;Well out of way of operating system
KERNAL1         EQU *
                STA VAL1
                AND #$0F
                STA VAL2
                LDA VAL1
                AND #$F0
                LSR
                LSR
                LSR
                LSR
                JSR CONV2
                STA VAL1
                LDA VAL2
                JSR CONV2
                STA VAL2
                RTS
CONV2           EQU *
                CMP #$10
                BCS AF
                CLC
                ADC #48
                RTS
AF              EQU *
                SEC
                SBC #9
                RTS
```

COMMAND CODE

This code is to provide all applications the access of the above operating system routine.

*Here is the example definition of an operating system routine which can be called by any application and is thus re-entrant this code must be loaded before the operating system is initialised. First define the jump to the code.

```
KERNAL1 = $2F00
```

*Below is an area to save the configuration. It must be in page zero as this code is also re-entrant.

```
SAVE1 = $FF
    ORG $0800
    PHA
    LDA $FF00
    STA SAVE1
    LDA #$3E  ;Operating system config
    STA $FF00
    PLA
    JSR KERNAL1
    PHA
    LDA SAVE1
    STA $FF00 ;Return to orig config
    PLA
    RTS
```

EXPANSION

Operating systems take many man hours of work to develop. They usually start off with a basic system that works and this is built on.
There are 2 different types of expansion. One is to develop the operating system and provide more facilities such as I/O control and software fdacilities such as program to program communication. Other expansions such as more operating system commands can also be considered. The other path is to develop applications to run under this operating system.
Both these udeas will be considered in turn outlining the approach.

EXPANSIONS TO THE OPERATING SYSTEM

There are so many different ways in which the system could be expanded and improved. I will list only a few.

TABLE IMPROVEMENTS

As described earlier in this series on storage and program tables, the table control is very simple and limits the system and how many storage entries exist. It would be better to chain table entries to effectively remove the limitations.

USER INTERFACING

My idea for this would be to define a screen area of 1K for each application that is run. The application can decide whether to use the screen or not. The maintenance of the screen would be via operating system calls eg

X = X position
Y = Y position
A = character

Call the routine to place this on application screen. Via the operating system, you can select which screen is displayed. A 'JUMP' key will switch the screens. This would be easy by just pointing the VIC II chip to the appropriate screen. For Colour however, you would have to swop the information into the appropriate area for colour. If an application was being displayed then you would have to update the usual screen area. The colour area used by the application and the colour RAM seen by VIC II. (ie $D800 - $DBFF) would also have to be updated. The operating system should also have it's own control screen. As for input, all user controlled input media such as keyboard and joystick should work as follows :-
Which ever application is currently being displayed (ie seen by VIC), then all user input is directed to that application. The application request input again via the operating system calls. No other application should get the information.This could be achieved by doing something like:-

Application

do until return pressed
get input via operating system
end

operating system input routine

Poll keyboard
if key pressed then
find out my program name
am I currently being displayed ?
if yes then return keyboard poll
end

You can locate which is the current program by looking in the system page 0 at area 'CURRENT'. A flag in the program table could indicate that this is the current one being displayed. You could even have a separate pointer to say which program is currently being displayed, and compare that with 'CURRENT'.

I/O CONTROL

Provision is required to control all the remaining I/O that is avialable. This should deal with the all the I/O registers within the I/O block. There are 2 ways in which this could be done. For each program, give them a set of shadow registers for all the I/O. when the program is given CPU time, place all the I/O information into the registers. When the program stops, copy all the information out again. to be used for next time. Obviously this idea cannot be used if you want to use real timers and get the correct time. (this idea could be used on sprites etc). This is called 'sharing resources'. Another idea is to dedicate a specific register to an application. Only that application can use that I/O register. This means that you get the real time, but you cannot use that register anywhere else. This is called 'dedicating resources'.

THAT CONCLUDES THIS MONTHS MONSTROUS TUTORIAL. NEXT MONTH WE COME TO THE END OF THE SERIES. WE WILL BE DISCUSSING VARIOUS THINGS SUCH AS AS; PROGRAM TO PROGRAM COMMUNICATIONS, INTERRUPTS , 80 COLUMN SUPPORT , TRACING AND DIAGNOSTICS AND A FEW OTHER GOODIES. I HOPE YOU ARE MANAGING TO STAY THE COURSE, AND AT THE SAME TIME LEARNING SOMETHING NEW ON THIS FASCINATING SUBJECT.

Commodore DISK·USER·

Lineage: 53p per word. (+VAT)
Semi display: £11.50 plus VAT per single column centimetre minimum 2cm.
Ring for information on series bookings/discounts.
All advertisements in this section must be prepaid.
Advertisements are accepted subject to the terms and conditions printed on the advertisement rate card (available on request).
Make cheques payable to Alphavite Publications Ltd.
Send your requirements to:
CLASSIFIED DEPARTMENT
ALPHAVITE PUBLICATIONS LTD., 20 POTTERS LANE, KILN FARM, MILTON KEYNES, MK11 3HF.

(0908) 569819

SOFTWARE

ONLY POOLS AND HORSES

FOOTBALL BOXFORM £14.95 cass £17.95 disk

Not a gimmicky plaything but a genuine statistical analysis of football form the sole objective of which is to improve the chances of winning on the pools or fixed odds. **Written by a pools expert the program has forecast over 50% more draws than would be expected by chance.** Homes, aways & draws are shown in order of merit and true odds given for every match. Merit tables show at a glance the teams currently in form and those having a lean spell. Separate AUSTRALIAN POOLS program included in the price.

RACING BOXFORM & HANDICAP WINNER £14.95 cass £17.95 disk

Two programmes for the price of one. Boxform weighs up a race by considering the many facets of form. Easily amended to suit the user's ideas. HANDICAP WINNER shows the value for money bets. Over 1000 winners every year and ON DERBY DAY 1990 THE PROGRAM GAVE THE WINNERS OF 7 OUT OF 9 HANDICAPS. 20/1 10/1 7/1 11/2 4/1 4/1 6/4. **PROBABLY THE BEST DAY'S RESULTS EVER ACHIEVED, WITH OR WITHOUT A COMPUTER.**

SPECIAL OFFER: All the above for £24.95 (disks £29.95) plus a FREE program to work out almost any bet. So good it's used by bookies.

Why pay inflated prices? **BOX**oft **CLEVER** the same – get the **BEST.** Advertised for five years in the sporting and computer press.

VISA

BOXOFT (CDU), 65 Allans Meadow, Neston, South Wirral, L64 9SQ
Cheque/P.O./Access/Visa Tel: 051-336-2668 (24 hrs)

Alphavite
PUBLICATIONS LIMITED

EDITORIAL ASSISTANT

YC Magazine is looking for a young, enthusiastic games fanatic to become a YOP Editorial Assistant.

The ideal applicant should have basic writing skills, would enjoy being wacky at shows, and must, above all else, enjoy playing computer games.

If you feel you could better the country's top C64 title, apply in writing to Rik Henderson – The Editor.

CLASSIFIED SALES EXECUTIVE

An excellent opportunity has arisen for a classified sales executive with at least 6 months experience to handle classified sales across 3 Commodore titles and 2 health magazines.

The position, based in Milton Keynes, offers an attractive lifestyle with competitive salary and commission package.

Please apply in writing to The Advertisement Manager.

20 Potters Lane, Kiln Farm, Milton Keynes MK11 3HF. Telephone: (0908) 569819 Fax: (0908) 260229

Lineage: **53p** per word. (+VAT)
Semi display: **£11.50** (+ VAT) per single column centimetre. Min size 2cm.
Series discounts available on request. All classified advertisements must be pre-paid.
There are no reimbursements for cancellations.
SEND TO CDU CLASSIFIED DEPARTMENT, ALPHAVITE PUBLICATIONS, 20 POTTERS LANE, KILN FARM, MILTON KEYNES, MK11 3HF.

I enclose my cheque/postal order for £..... for..... insertions, made payable to Alphavite Publications.
(Delete as necessary)
PLEASE DEBIT MY ACCESS/VISA CARD NO: | | | | | | | | | | | | | | | | |
EXP. DATE. .....
£ .....FOR..... INSERTIONS
Name .....
Address ..... Postcode .....
Daytime Tel No ..... Signature ..... Date .....

❑ FOR SALE ❑ SOFTWARE ❑ SPECIAL OFFERS ❑ REPAIRS ❑ HARDWARE ❑ DISKS

...it's dynamite!

POWER CARTRIDGE

FOR YOUR COMMODORE 64 / 128

"...unbelievable value for money" ZZAPP! Dec 89

- POWER TOOLKIT
- POWER MONITOR
- TAPE & DISK TURBO
- PRINTERTOOL
- POWER RESET
- TOTAL BACKUP

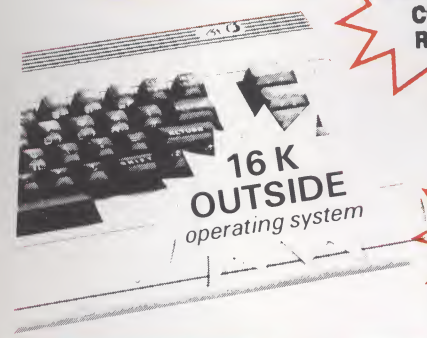
..."Money well spent" YC/CDU Jan 90

42 Pg Manual - "Damned Good Handbook" CCI Jan 90

SO MUCH FOR SO LITTLE

AVAILABLE FROM ALL GOOD COMPUTER RETAILERS

TRIED AND TESTED - OVER 100,000 SOLD IN EUROPE



..."highly recommended for C64 users" CCI Jan 90

YOU WILL WONDER HOW YOU EVER MANAGED WITHOUT IT

POWER TOOLKIT

A powerful BASIC-Toolkit (Additional helpful commands) that considerably simplifies programming and debugging.

AUTO	HARDCAT	RENUMBER
AUDIO	HARDCOPY	REPEAT
COLOR	HEX$	SAFE
DEEK	INFO	TRACE
DELETE	KEY	UNNEW
DOKE	PAUSE	QUIT
DUMP	PLIST	MONITOR
FIND	ILOAD	BLOAD

RENUMBER : *Also modifies all the GOTO's GOSUB's etc. Allows part of a program to be renumbered or displaced.*
PSET : *Set up of printer type.*
HARDCAT : *Prints out Directory.*

The toolkit commands can be used in your programs.

DISK TOOL

Using POWER CARTRIDGE you can load up to 6 times faster from disk.
The Disk commands can be used in your own programs.

DLOAD	DVERIFY	DIR
DSAVE	MERGE	DEVICE
DISK		

MERGE : *Two BASIC programs can be merged into one.*
DISK : *With DISK you can send commands directly to your disk.*

TAPE TOOL

Using POWER CARTRIDGE you can work up to 10 times faster with your data recorder. The Tape commands can be used in your own programs.

LOAD	SAVE	VERIFY
MERGE	AUDIO	

POWERMON

A powerful machine language monitor that is readily available and leaves all of your Commodore memory available for programming.
Also works in BASIC-ROM, KERNAL and I/O areas.

A ASSEMBLE	I INTERPRET	S SAVE
C COMPARE	J JUMP	T TRANSFER
D DIS-ASSEMBLE	L LOAD	V VERIFY
	M MEMORY	W WALK
F FILL	P PRINT	X EXIT
G GO	R REGISTER	$ DIRECTORY DOS Commands
H HUNT		

PRINTERTOOL

The POWER CARTRIDGE contains a very effective Printer-Interface, that self detects if a printer is connected to the Serial Bus or User-Port.
It will print all Commodore characters on Epson and compatible printers.
The printer-interface has a variety of set-up possibilities. It can produce HARDCOPY of screens not only on Serial printers (MPS801, 802, 803 etc) but also on Centronic printers (EPSON, STAR, CITIZEN, PANASONIC, etc).
The HARDCOPY function automatically distingishes between HIRES and LORES. Multi-colour graphics are converted into shades of grey. The PSET functions allow you to decide on Large/Small and Normal/Inverse printing.
The printer PSET functions are:

PSET 0 - *Self detection Serial/Centronics.*
PSET 1 - *EPSON mode only.*
PSET 2 - *SMITH-CORONA mode only.*
PSET 3 - *Turns the printing 90 degrees!!*
PSET 4 - *HARDCOPY setting for MPS802/1526.*

PSET B - *Bit-image mode.*
PSET C - *Setting Lower/Upper case and sending Control Codes.*
PSET T - *All characters are printed in an unmodified state.*
PSET U - *Runs a Serial printer and leaves the User-port available.*
PSET Sx - *Sets the Secondary address for HARDCOPY with Serial Bus.*
PSET L1 - *Adds a line-feed, CHR$ (10), after every line.*
PSET L0 - *Switches PSET L1 off*

POWER RESET

On the back of the POWER CARTRIDGE there is a Reset Button. Pressing this button makes a SPECIAL MENU appear on the screen.
This function will work with many programmes.

CONTINUE - *Allows you to return to your program.*
BASIC - *Return to BASIC.*
RESET - *Normal RESET.*
TOTAL BACKUP DISK - *Saves the contents of the memory onto a Disk. The program can be reloaded later with BLOAD followed by CONTINUE.*
RESET ALL - *RESET of any program.*
TOTAL BACKUP TAPE - *As BACKUP DISK but to TAPE.*
HARDCOPY - *At any moment, prints out a Hardcopy of the screen. Using CONTINUE afterwards you can return to the program.*
MONITOR - *Takes you into the Machine language Monitor.*

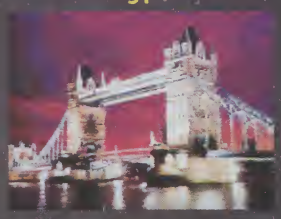
BDL
Bitcon Devices Ltd

88 BEWICK ROAD
GATESHEAD
TYNE AND WEAR
NE8 1RS
ENGLAND

Tel: 091 490 1975 and 490 1919 Fax 091 490 1918
To order: Access/Visa welcome - Cheques or P/O payable to BDL
Price: £16.99 incl. VAT.
UK orders add £1.20 post/pack total - £18.19 incl. VAT.
Europe orders add £2.50. Overseas add £3.50
Scandinavian Mail Order and Trade enquiries to: Bhiab Elektronik, Box 216, Norrtalje 76123, SWEDEN. Tel: + 46 176 18425 Fax: 176 18401
TRADE AND EXPORT ENQUIRIES WELCOME

AMIGA

ST



PC

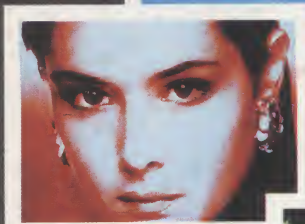
Actual Digitised colour screen - shots

VIDEO COLOUR SPLITTER

£69.95 Inc Vat

Vidi-RGB is an electronic filter which takes a colour video signal and separates it into the three primary colours (Red, Green and Blue) allowing each to be digitised.

Ideal for use with Vidi-Chrome & Frame Grabber or Digi-View Gold (By Newtek).



★ For use with colour Digitisers replacing conventional Filter sets.

★ Our Vidi - Chrome switches Vidi - RGB automatically grabbing full colour pictures in less than one second.

★ Digitise full colour images direct from home VCR (must have perfect freeze frame)

★ Digitise outstanding colour pictures direct from Canon's new Still Video Camera (an example shown on cover)

★ Manual switching for maximum flexibility.

★ Fully compatible with Digi - View Gold.

All these pictures are **actual unretouched screen-shots** illustrating the sequence of creating a full colour image using Vidi RGB, Vidi-Chrome and Vidi Amiga.

Limited

6 Fairbairn Road,
Kirkton North,
Livingston,
Scotland,
EH54 6TS.

Tel: 0506-414631
Fax: 0506-414634